ProGIP: Protecting Gradient-based Input Perturbation Approaches for Out-of-distribution Detection From Soft Errors

by

Sumedh Shridhar Joshi

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2024 by the
Graduate Supervisory Committee:

Aviral Shrivastava, Chair
Aman Arora
Hokeun Kim

ARIZONA STATE UNIVERSITY

December 2024

ABSTRACT

Undetected out-of-distribution (OOD) inputs, the representative non-malicious threat, pose a significant menace to the reliability of deep learning models as they can lead to unexpected behaviors during inference. Several studies proposed novel OOD input detection methods and proved their effectiveness. However, soft errors, another representative non-malicious threat, can impact both the classification results of neural network models (in-distribution) and ID/OOD detection results of OOD detection solutions. Therefore, protecting OOD detection solutions from soft errors is crucial to ensure the reliability of neural networks. Still, none of the existing solutions can detect and distinguish soft errors from OOD inputs. To provide a resilient OOD detection solution against soft errors, this thesis analyzes the effect of soft errors on neural network models with gradient-based input perturbation (GIP) approaches, which are representative OOD detection solutions. Building on my analysis, I propose ProGIP, which incorporates two software-level range-based fault detectors to protect all execution phases of GIP approaches, including two forward and one backward pass. My ProGIP solution enables GIP approaches to distinguish between ID, OOD, and fault- affected inferences, detecting nearly 99% of critical faults while introducing a negligible runtime overhead of less than 1%.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my thesis supervisor, Dr. Aviral Shrivastava, whose unwavering guidance and support have been instrumental throughout my research journey. Your expertise and encouragement have inspired me to push the boundaries of my work. I am also deeply thankful to Dr. Hwisoo So for his invaluable guidance and support, which has greatly shaped my understanding of the subject. My sincere appreciation extends to my thesis committee members, Dr. Aman Arora and Dr. Hokeun Kim, whose insightful feedback and constructive criticism have significantly enhanced the quality of this thesis. Your contributions have been vital in enriching my research and deepening my analysis. Thank you all for your dedication and support; I am truly grateful for your mentorship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

With the remarkable success of deep neural networks, machine learning has become a pivotal advancement to modern computing systems. Safety-critical systems such as autonomous driving [11] and disease diagnosis [19] adopt machine learning techniques as the core algorithm. However, the potential consequences of malfunctions or unexpected behavior in such systems are catastrophic [4]. In real-world environments, non-malicious threats can severely undermine the reliability of machine learning models, even in the absence of malicious attacks. As a result, ensuring the reliability of machine learning models against non-malicious threats has become a paramount design concern for AI-based safety-critical systems. [42]

Out-of-distribution (OOD) data is a major non-malicious threat to the machine learning system. Since machine learning models are designed to learn a training dataset in a particular distribution, they lack the proper information to deal with OOD data, which deviates significantly from in-distribution data (ID). Therefore, correct inference of OOD inputs is difficult, or in some cases, even impossible. A better approach is to detect OOD data and reject such unfamiliar cases. Such an approach prevents the machine learning models from behaving abnormally due to unreliable inference results of OOD data [39]. For instance, if an autonomous driving system encounters objects never encountered during training, it should recognize the anomaly and deliver a warning [46, 4].

The representative OOD detection category is gradient-based input perturbation (GIP) [27, 16, 22, 13, 44, 50, 41, 48, 18], which utilizes the gradients to perturb the input to maximize the confidence gap between ID and OOD inputs. However, such

solutions are vulnerable to **soft errors**, another major threat to reliability. A soft error is a transient fault resulting in a temporary bit-flip error of the transistor, induced by external sources such as alpha particles, thermal neutrons, or cosmic rays [29]. My experiments show that soft errors in neural networks can induce incorrect classification results and affect the ID/OOD detection results of GIP solutions. In neural networks, these bit-flips on execution units can alter neuron results. Given that GIP methods require precise computation of gradients and confidence scores, soft errors can severely affect their effectiveness by introducing miscalculations.

No existing solutions provide comprehensive reliability against both OOD data and soft errors. Few solutions can potentially detect both threats as outliers, but they cannot distinguish whether the detected outlier is a case of OOD data or soft error. Distinguishing between OOD data and soft errors is essential for developing specific countermeasures for different threats. For example, operators of machine learning applications can collect detected OOD inputs as potential learning resources for future use. [47, 32, 35]. On the other hand, since voltage and frequency affect the soft error rate [7], system designers can monitor soft error occurrences to adjust dynamic voltage and frequency scaling (DVFS) settings [38]. Such threat-specific actions are impossible without a method of distinction between different threats.

Combining GIP approaches and soft error detection solutions for comprehensive reliability is challenging due to the structure of GIP approaches. GIP approaches [27, 16, 22, 13, 44, 41, 48, 18] require one pair of forward and backward passes to obtain the gradients for input perturbation and an extra forward pass to generate the confidence scores of the perturbed input. Due to the need for multiple executions, GIP solutions are extremely sensitive to additional runtime overhead incurred by applying soft error detection solutions. Furthermore, most soft error detection solutions only provide detection methodologies for the forward pass, although GIP solutions also

require a backward pass.

To achieve comprehensive reliability against non-malicious threats, I propose *ProGIP* (Protecting Gradient-based Input Perturbation), which protects GIP approaches for OOD detection against soft errors with minimum software-level detectors based on the range-based checkers [2, 9], to distinguish ID, OOD, and fault-affected cases. Inspired by the observation in previous studies [2, 9] that faults on the high-order bits in the model mainly contribute to the failures, this thesis analyzes the effects and symptoms of high-order bit-flips in ID/OOD detection results along with classification results. Based on the analysis, I have developed *ProGIP*, which inserts two software-level range-based checkers to detect the majority of high-order bit-flips in all three passes of GIP approaches; one checker right after the backward pass to cover the first forward and backward passes, and another checker after the second forward pass to cover that pass. The contribution of this thesis is as follows:

- I analyze the impact of soft errors on neural networks using gradient-based input perturbation (GIP) approaches, identifying vulnerabilities in existing out-of-distribution (OOD) detection methods.

- I propose *ProGIP*, a holistic reliability solution that integrates two fault detectors to protect both forward and backward passes in GIP approaches, effectively distinguishing between OOD data and soft errors.

- My method detects 98.76% of critical failures on neural network models with GIP solution with only 0.81% runtime overhead, ensuring comprehensive reliability for deep learning models against non-malicious threats.

Chapter 2

RELATED WORK

2.1 Gradient-based input perturbation (GIP) solutions for out-of-distribution (OOD) detection

A fundamental strategy to distinguish ID and OOD data is to score the confidence of inference results based on the softmax values, based on the observation that the highest softmax results of ID inferences tend to be higher than the ones of OOD inferences [14]. No existing single method can consistently outperform others across benchmarks, and their performance ranking is different from one dataset to another [49]. In this thesis, I focus on the GIP approach, which is one of the most representative approaches in OOD detection categories. For the sake of simplicity, instead of introducing all GIP solutions [27, 16, 22, 13, 44, 41, 48, 18] that share the similar structure, this thesis mainly discusses two representative GIP solutions, ODIN [27] and Mahalanobis [22].

The main goal of the GIP approach is to maximize the confidence score gaps between ID and OOD by perturbing the input based on the gradient so that GIP solutions can distinguish ID and OOD more effectively. Figure 2.1 shows the high-level view of GIP solutions. In this figure, the GIP approach first generates the perturbed input with an input perturbation process, which includes first forward and backward passes to generate gradients. Inspired by the fast gradient sign method [10], the input perturbation in this process maximizes the confidence gap between ID and OOD inputs using Equation 2.1.

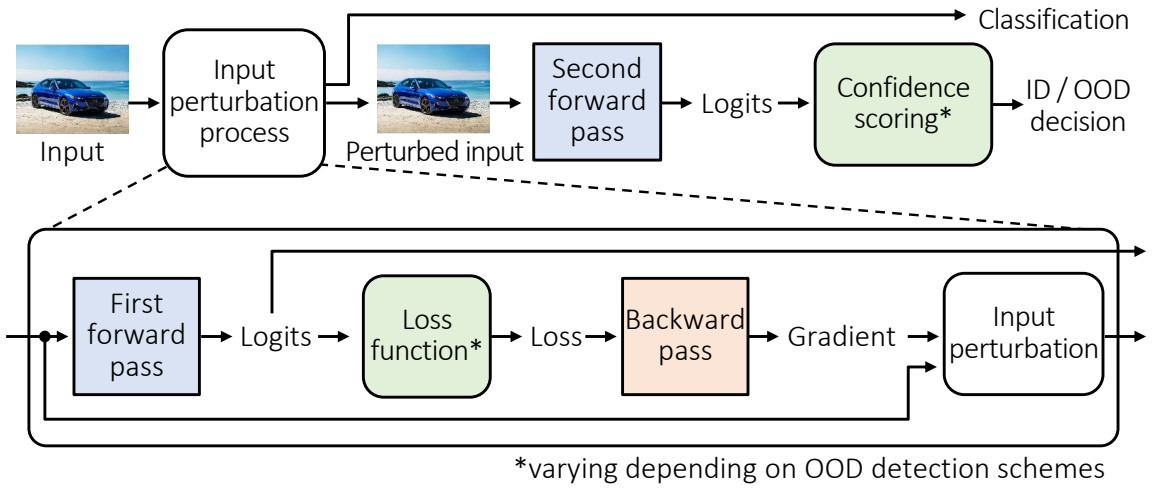

Figure 2.1: Gradient-based input perturbation (GIP) solutions process first forward and backward passes to perturb the input, and second forward pass to distinguish ID and OOD.

$$\tilde{x} = x - \epsilon * sign(-\nabla_x score(x)) \tag{2.1}$$

In Equation 2.1, x and $\tilde{x}$ represent the original and perturbed inputs, respectively. $(-\nabla_x score(x))$ represents the gradient of the scoring function with respect to sample input, where the scoring function varies based on solutions. *sign* function preserves the sign of the gradient values regardless of their magnitudes; 1 if the value is positive, and -1 if the value is negative. The theoretical background for the gradient-based input perturbation is well discussed in ODIN [27].

After the input perturbation process, GIP solutions process a second forward pass with the perturbed input and score the confidence with logits. Based on the confidence scores, GIP solutions judge whether the input is ID or OOD. The confidence scoring function in the GIP approach varies depending on GIP solutions. ODIN [27] uses maximum softmax value with temperature scaling, which is utilized to distill the knowledge in a neural network [15] or to calibrate the confidence [12]. On the other hand, Mahalanobis [22] calculates the Mahalanobis distance between the logits

generated with perturbed input and the mean of logits generated with ID inputs for each class. Mahalanobis utilizes the negative of the maximum Mahalanobis distance value as a confidence score.

In both solutions, the confidence scores of ID inputs tend to be higher than those of OOD inputs. Therefore, both solutions distinguishes between ID and OOD by equation 2.2.

$$g(x) = \begin{cases} OOD & if \quad score(\tilde{x}) \leq \delta \\ ID & if \quad score(\tilde{x}) > \delta \end{cases} \tag{2.2}$$

In Equation 2.2, δ represents the threshold to classify ID/OOD, $g(x)$ represents the final confidence score of the respective methods, x and $\tilde{x}$ represent the original and perturbed inputs, respectively. If the confidence is higher than the threshold δ, then the input is classified as ID, else it is classified as OOD. A higher threshold increases the chances of correctly detecting OOD, but it can also increase the chances of ID inputs with lower confidence scores being categorized as OOD. For the implementation of ODIN and Mahalanobis, I consider ID as positive and OOD as negative according to the metric of ODIN [27], and set the threshold at 95% true positive rate (TPR), i.e., the threshold that misidentifies 5% of IDs as OOD and correctly identifies 95% of IDs as ID.

Note that GIP solutions in Figure 2.1 do not utilize the classification results of the second forward pass with perturbed input since the input perturbation may alter the classification results. Instead, they can utilize the result of the first forward pass with the original input as the classification result.

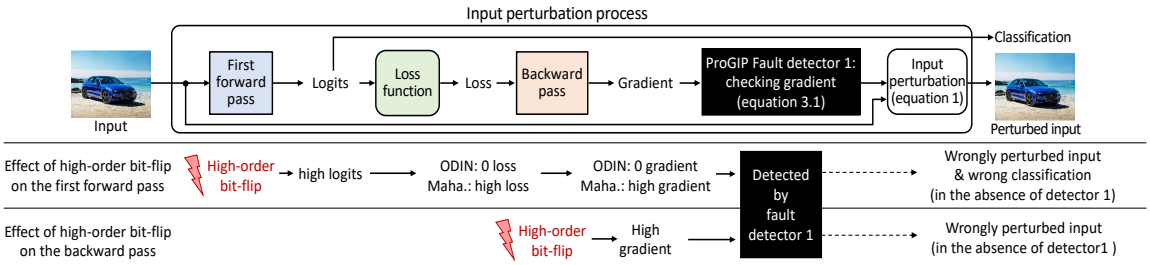

Figure 2.2: High-order bit-flips on the first forward and backward passes in GIP solutions induce extremely high or near zero gradients, which is tricky to observe in error-free inferences. The first fault detector of *ProGIP* checks the gradient based on this observation to cover high-order bit-flips on such passes.

2.2 Soft error mitigation for neural networks

Traditional redundancy schemes against soft errors such as dual modular redundancy (DMR) are hard to apply to neural networks due to their hardware cost or runtime overhead. Therefore, soft error protections for neural networks focus on efficient redundancy. For example, algorithm-based fault tolerance (ABFT) solutions [31, 1] exploit mathematical properties of matrix multiplication operations to introduce efficient checksums.

Another solution [37] trains a small network that can detect and correct fault-affected executions of the target network, based on the fault-free and fault-affected execution traces. Range-based solutions [2, 9] insert checkers between layers of the target network to detect abnormally high values affected by faults.

The range-based studies observed that faults on high-order bits contribute to the majority of soft-error-induced failures of neural networks [2, 9], and such large deviations can be easily detected since they induce significantly large values in layer outputs which rarely appear in the absence of the faults [23, 2, 9]. For example, a single-bit fault on the highest-order exponent bit in the IEEE-754 floating point representation can change a value from 0.5 to over 10^{38}. Such large deviations can

change the final classification result of the model regardless of the original small values in the network.

Detecting soft errors in OOD detection solutions is crucial since soft errors in neural networks not only affect the classification result but also can change the ID/OOD detection result. Only a few solutions [36, 8] have the potential to provide holistic reliability against non-malicious threats including both hardware faults and OOD inputs, by considering soft error (or permanent fault) and OOD as an anomaly. However, such solutions cannot distinguish faults and OOD inputs, which is crucial for system designers who want to establish appropriate mitigation solutions for each anomaly type.

Chapter 3

METHODOLOGY

In this chapter, I propose *ProGIP*, an efficient mechanism to detect the majority of critical faults in GIP solutions to prevent two types of failures on GIP solutions. **i) Classification failure:** A neural network model with a GIP solution can correctly classify the input in the absence of soft errors, but a soft error causes the model to misclassify the input. **ii) ID/OOD detection failure:** A neural network model with a GIP solution can correctly distinguish the input as ID or OOD in the absence of soft errors, but a soft error causes the model to misclassify an ID as OOD (ID to OOD) or vice versa (OOD to ID).

Several studies have demonstrated that the majority of soft-error-induced failures in neural networks are caused by high-order bit-flips [23, 2, 9]. Motivated by these observations, *ProGIP* analyzes how high-order bit-flips on the three execution passes of GIP solutions induce these two types of failures and what visible symptoms these bit-flips produce, and places two fault detectors that can protect all three execution passes of GIP solutions. The first fault detector in 3.1 detects abnormal gradient values to protect the first forward and backward passes. The second fault detector in 3.2 detects abnormal confidence scores to protect the second forward pass.

3.1 First fault detector for first forward and backward passes

Figure 2.2 illustrates the effects of high-order bit-flips on the first forward and backward passes of GIP solutions, and the corresponding first fault detector of *ProGIP*. A high-order bit-flip on the first forward pass results in unusually high logits. These

logits are used for classification and fed to the backward pass to generate gradients for input perturbation. Therefore, such incorrect logits can directly induce classification failures, and also can indirectly induce ID/OOD detection failures by affecting the gradient for the input perturbation.

The effect of high-value logits on the gradient varies depending on the loss functions in ODIN and Mahalanobis. ODIN employs softmax-based confidence scoring and cross-entropy loss. When logits are high, the cross-entropy loss approaches zero, leading to an almost zero gradient in the backward pass. On the other hand, Mahalanobis uses distance-based confidence scoring, calculating the Mahalanobis distance using the logits of the input ($f(x)$) and the mean and covariance of logits of ID samples for a class ($\hat{\mu}_c$ and $\hat{\Sigma}_c$, respectively). Then, the confidence score is defined as the closest Mahalanobis distance among classes, i.e., $max_c\{-(f(x)-\hat{\mu}_c)^T\hat{\Sigma}_c^{-1}(f(x)-\hat{\mu}_c)\}$. In this equation, as the logits $f(x)$ increase, the confidence scores decrease since the confidence is the negative of distance. Consequently, the loss will increase, resulting in a higher gradient in the backward pass.

High-order bit-flips on the backward pass directly result in a high value-gradient as illustrated in Figure 2.2, regardless of the types of loss functions. Similar to the high-order bit-flips on the first forward pass of Mahalanobis, such high gradient values can indirectly induce ID/OOD detection failures by affecting the input perturbation, but bit-flips in the second forward pass do not affect classification results.

The symptom of high-order bit-flips on the first forward and backward passes found by the above analysis is extremely high or near-zero gradient values. However, such high deviations disappear after the input perturbation, since the input perturbation with equation 2.1 only utilizes the sign of the gradient and disregards the magnitude. Note that this does not mean that the input perturbation process masks the effects of high-order bit-flips entirely, since the signs of gradient values can also

be affected by faults.

Therefore, as illustrated by the black box in Figure 2.2, $ProGIP$ places the first fault detector right before the input perturbation, which can be implemented at the software level with equation 3.1.

$$check(grad) = \begin{cases} Faulty & if \quad max(abs(grad)) \approx 0, \\ Faulty & if \quad max(abs(grad)) > \delta_{F1}, \\ Fine & otherwise. \end{cases} \tag{3.1}$$

In Equation 3.1, $grad$ indicates the gradient from the backward pass of GIP solutions, and δ_{F1} indicates the fault detection threshold for the first fault detector of $ProGIP$. For Equation 3.1, I considered any value less than 10^{-11} as near-zero for the first comparison. Note that since high-order bit-flips never induce near-zero gradients in Mahalanobis, the first fault detector can skip the comparison against zero in Mahalanobis. To select the fault detection threshold δ_{F1}, I profile the $max(abs(grad))$ values from the inferences with the ID training dataset and find the maximum value of $max(abs(grad))$. To prevent naturally large gradients from the test dataset from causing false fault detection alarms, I conservatively add a 200% margin, i.e., triple it, to the selected maximum value.

3.2 Second detector for second forward pass

Figure 3.1 illustrates the effects of high-order bit-flips on the second forward pass of GIP solutions, and the corresponding second fault detector of $ProGIP$. High-order bit-flips in the second forward pass induce extremely high logits. Since the confidence scoring functions of GIP solutions utilize logits from the second forward pass, such faults can directly induce ID/OOD detection failures. Specifically, the unusually high logits result in different scores based on the confidence scoring functions of ODIN and

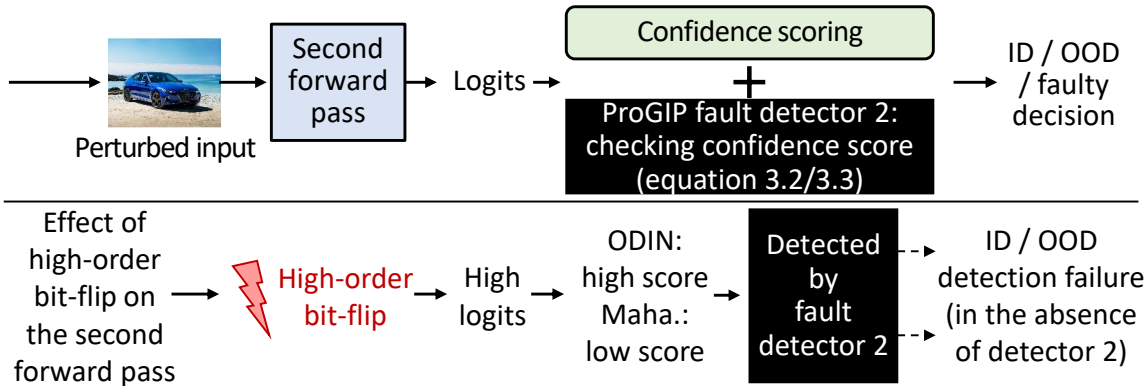

Figure 3.1: High-order bit-flips on the second forward pass of GIP solutions induce extremely high or low confidence scores compared to fault-free inferences. Based on this observation, the second fault detector of *ProGIP* checks the confidence score to detect high-order bit-flips on the second forward pass.

Mahalanobis.

The softmax-based confidence scoring of ODIN adopts drastic temperature scaling. The fault-free confidence scores of ODIN for both ID and OOD inputs are relatively small, typically near the reciprocal of the number of classes. In contrast, high logits resulting from high-order bit-flips during the second forward pass lead to extremely high confidence scores in ODIN, even higher than the fault-free scores for ID inputs. Therefore, if a high-order bit-flip affects the second forward pass of ODIN during the inference of an OOD input, the binary classification in Equation 2.2 will classify the input as ID, resulting in an ID/OOD detection failure (OOD to ID). For example, in ResNet with ODIN, I observed fault-affected confidence scores reaching values near 1, while the average confidence scores of ID and OOD inferences were 0.10104 and 0.10082, respectively.

On the other hand, fault-induced high logits in a neural network with Mahalanobis can produce low confidence scores: that is, scores with a high absolute value but negative. This happens because such high logits are extremely distant from the distribution of fault-free outputs. If a high-order bit-flip affects the second forward

pass of Mahalanobis during the inference of an ID input, the confidence score would decrease significantly. Therefore, the binary classification will classify the input as OOD, resulting in an ID/OOD detection failure (ID to OOD). For example, in ResNet with Mahalanobis, I observed a fault-affected confidence score of $-2.2e^{11}$, while the average confidence scores of ID and OOD inferences were -172.07 and -125.95, respectively.

The symptom of high-order bit-flips on the second forward pass found by the above analyses is an extremely high confidence score in the case of ODIN and an extremely low confidence score in the case of Mahalanobis. Therefore, as illustrated by the black box in Figure 3.1, $ProGIP$ places the software-level second fault detector, which jointly checks the logits with the confidence scoring function to distinguish ID, OOD, and faulty inferences. Since symptoms vary depending on the confidence scoring function, $ProGIP$ also provides different fault detectors for ODIN and Mahalanobis. Equation 3.2 shows how the second fault detector with the scoring function of ODIN distinguishes between ID, OOD, and faulty inferences.

$$g'_O(x) = \begin{cases} OOD & if \quad score_O(\tilde{x}) \leq \delta_{OOD}, \\ ID & if \quad \delta_{OOD} < score_O(\tilde{x}) \leq \delta_{F2}, \\ Faulty & if \quad score_O(\tilde{x}) > \delta_{F2}. \end{cases} \tag{3.2}$$

In equation 3.2, x represents the original input, and $\tilde{x}$ represents the perturbed input. g'_O represents the modified ODIN OOD detector with the second fault detector of $ProGIP$. This detector compares the confidence score of ODIN ($score_O$), i.e., maximum softmax with the temperature scaled logits from the second forward pass, against the ID/OOD threshold δ_{OOD} and fault detection threshold δ_{F2}. Since high-order bit-flips in the second forward pass of ODIN mostly result in high confidence, the second detector of $ProGIP$ considers the inference as faulty if $score_O(\tilde{X})$ is higher

than the threshold δ_{F2}.

Equation 3.3 shows how the second fault detector with the scoring function of Mahalanobis distinguishes between ID, OOD, and faulty inferences.

$$g'_M(x) = \begin{cases} Fault & if \quad score_M(\tilde{x}) \leq \delta_{F2}, \\ OOD & if \quad \delta_{F2} < score_M(\tilde{x}) \leq \delta_{OOD}, \\ ID & if \quad \delta_{OOD} < score_M(\tilde{x}). \end{cases} \tag{3.3}$$

In equation 3.3, g'_M and $score_M$ represent the modified Mahalanobis OOD detection and the confidence scoring function of Mahalanobis, respectively. High-order bit-flips in the second forward pass of Mahalanobis usually result in low confidence, and therefore the second detector of $ProGIP$ detects the faulty inference if $score_M(\tilde{X})$ is loIr than the threshold δ_{F2}.

As discussed in Chapter 2, I select ID/OOD threshold δ_{OOD} at 95% TPR for ID/OOD classification. To select the fault detection threshold δ_{F2}, I profile the confidence scores with the ID training dataset without injecting faults and add a 200% margin, similar to the threshold for the first fault detector in 3.1. For δ_{F2} of ODIN, I first find the logits that result in the maximum confidence score during the training phase and then increase it by 200%. For δ_{F2} of Mahalanobis, I found the minimum of the score during the training phase and tripled it.

Chapter 4

EXPERIMENTAL SETUP

I have conducted runtime measurement and fault injection experiments to evaluate the efficiency and fault coverage of *ProGIP*.

Network, dataset, and OOD detection: I adopt DenseNet-BC [17] and ResNet34 [50] trained to classify CIFAR-10 dataset. I use the pre-trained DenseNet-BC provided by the official implementation of ODIN [26], which trains with a growth rate of 12 for 300 epochs with a Iight decay of e^{-4}. For ResNet34, I trained it for 200 epochs with a Iight decay of $5e^{-4}$. Both models use a Stochastic Gradient Descent (SGD) optimizer and a learning rate of 0.1 divided by 10 at 50% and 75% of the total number of training epochs. For both model, I apply ODIN [27] and Mahalanobis [22] by adopting their respective official implementation on GitHub [26, 21]. Table 4.1 summarizes the classification and ID/OOD detection accuracies for the models and GIP solutions.

Soft error detection: I implement two software-level *ProGIP* checkers in 3.1

Table 4.1: Network and OOD detection information

Network	OOD detection	ID Classification accuracy	AUROC for TPR vs FPR curve	FPR at 95% TPR
2*DenseNet-BC	ODIN	2*95.19%	98.40%	8.0%
	Mahanalobis		96.40%	15.0%
2*ResNet-34	ODIN	2*94.61%	90.30%	39.2%
	Mahanalobis		93.00%	35.2%

and 3.2 into the implementation of ODIN and Mahalanobis. As discussed in 3.1 and 3.2, the fault detection thresholds are selected based on the profiling with the ID training dataset. Since the comparison operations in PyTorch may behave differently in cases of not-a-number (NaN) or infinite numbers, I additionally checked for NaN and infinite numbers for the two checkers of *ProGIP*.

There is no existing solution to provide soft error detection for GIP solutions. To compare the efficiency and effectiveness of *ProGIP*, I additionally implement a detection-only Ranger [2] with GIP solutions, which checks the abnormal values of all activation function layers and following max pool, average pool, and reshape layers, via built-in hook methods of PyTorch. The Original Ranger also tries to correct faults by changing the abnormal values, but my detection-only Ranger is implemented to only provide a fault detection alarm. Similar to the *ProGIP* implementation, I added NaN and infinite number checking for the last layers of the execution passes of GIP solutions with Ranger.

Fault injection setup: I assume that ECC and parity can effectively cover the faults in memory [34], and therefore I aim to mimic soft errors in the datapath. Several prior studies flipped bits in the outputs of the layers in a neural network as virtual soft errors in the datapath [2, 23, 3, 37, 24]. According to the prior studies, I implemented bit-flips in the outputs of layers[1] via hook methods built-in PyTorch.

For one fault injection trial, I randomly select a layer in one execution pass. Then, I execute an inference of the GIP solution with both *ProGIP* and the detection-only Ranger. Just before executing the selected pass, I add a fault injection hook and execute the pass so that the hook will directly flip a random bit in the output of the selected layer. After executing the selected pass, I remove the added hook and continue the remaining execution.

[1] https://github.com/hidden_due_to_the_blind_review

Table 4.2: Runtime measurement results

2*Network	2* OOD detection	Normalized execution time	
		Detection-only Ranger	*ProGIP*
2*DenseNet-BC	ODIN	264.79%	100.74%
	Mahanalobis	259.01%	100.68%
2*ResNet-34	ODIN	245.48%	101.24%
	Mahanalobis	234.17%	100.56%
Average		250.86%	100.81%

For the fault injection run, I collect classification, ID/OOD detection, and soft error detection results from both protections. To identify the classification and ID/OOD detection failures, I also execute the inference with the same input and random seeds without injecting fault and compare the results between fault injection and fault-free runs. I have repeated a total of 2.4 million fault injections; 100,000 fault injection runs $\times$ 2 networks $\times$ 2 OOD detections $\times$ 2 types of dataset (ID or OOD) $\times$ 3 execution passes.

Runtime measurement: I measured execution times of ODIN and Mahalanobis, without soft error protection, with the detection-only Ranger, and with *ProGIP* on Google Colab with NVIDIA T4 GPU. I executed inferences with 1,000 ID and 1,000 OOD inputs without batch as one runtime measurement iteration and repeated 20 iterations. I computed the mean of execution times for each version by excluding the top and bottom 10% of iterations.

Table 4.3: Summary of the fault injection experiments

	Fault injection run	Originally correct run*	Classification failure	ID/OOD detection failure
Unprotected	3*2,400,000	3*2,004,945	4,318	5989
Detection-only Ranger			12	133
ProGIP			17	111

* The execution that can produce correct classification and ID/OOD results if a fault is not injected.

Chapter 5

EXPERIMENTAL RESULTS

Table 4.2 shows the execution times of ODIN and Mahalanobis with the detection-only Ranger or *ProGIP*, normalized by the execution times of unprotected ODIN and Mahalanobis. The detection-only Ranger induces 150.86% runtime overhead on average. I observed that the backward hook methods for every ReLU layer induce severe overhead. Note that it is usual to custom backward pass via hooks in PyTorch [25, 20, 30]. On the other hand, *ProGIP* only induced 0.81% runtime overhead, since it only inserts two checkers.

Table 4.3 summarizes the fault injection results on ODIN and Mahalanobis, without protection, with the detection-only Ranger, and with *ProGIP*. In this table, I only counted classification and ID/OOD failures for the originally correct runs, i.e.,

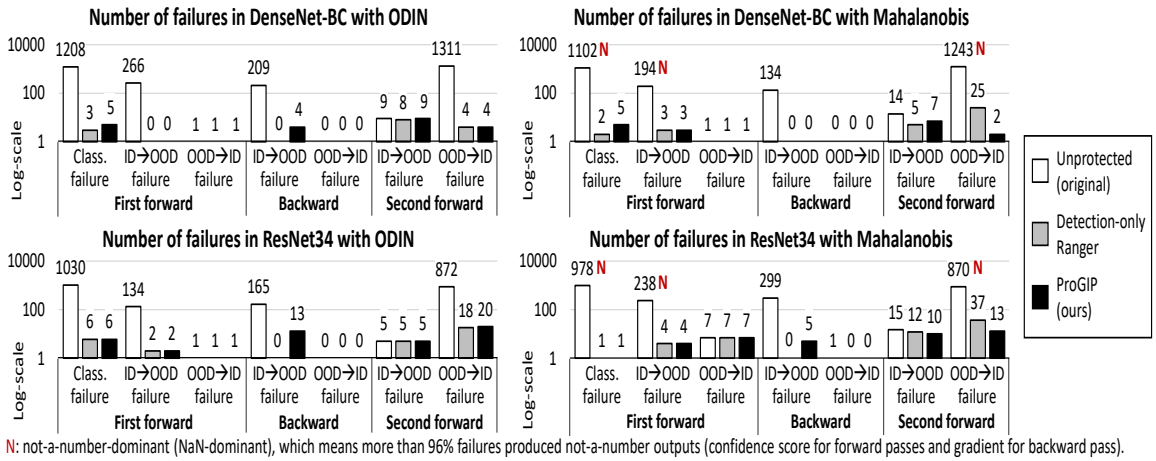

Figure 5.1: The detailed fault injection results show the coverage of *ProGIP* outperforms the one of the detection-only Ranger for OOD to ID detection failures induced by faults on the second forward pass of Mahalanobis, and comparable to the one of the detection-only Ranger for other cases.

the execution can produce correct results if a fault is not injected. In other words, if the fault-free run with the same input and seed failed to produce either correct classification or ID/OOD detection failure, I did not count classification and ID/OOD detection failures from such failed executions. In Table 4.3, unprotected ODIN and Mahalanobis encountered a total of 4,318 and 5,989 classification and ID/OOD detection failures, respectively. The detection-only Ranger encountered 12 classification and 133 ID/OOD detection failures. Finally, ProGIP encountered 17 classification and 111 ID/OOD detection failures, achieving comparable or even better failure coverage (98.76%) compared to the detection-only ranger with minimum overhead.

Figure 5.1 shows the detailed fault injection results. In this figure, white bars, grey bars, and black bars represent the number of failures in the unprotected version, the detection-only ranger version, and *ProGIP* version, respectively. The number of failures in unprotected versions marked with red N means not-a-number-dominant (NaN-dominant), i.e., more than 96% failures produce NaN value output from at least one forward or backward pass. On the other hand, all of the other numbers of failures in unprotected versions include around from 0% to 15.3% NaN cases.

The scoring function of Mahalanobis computes $max_c\{-(f(x)-\hat{\mu_c})^T\hat{\Sigma}_c^{-1}(f(x)-\hat{\mu_c})\}$ where $f(x)$ is logits and $\hat{\Sigma}_c^{-1}$ and $\hat{\mu}_c$ are the mean and covariance of logits of ID samples for a class c. In this equation, high logits ($f(x)$) can easily produce NaN in PyTorch. Such NaNs in my PyTorch environment always make the classification result 0 (first label). Also, comparisons in PyTorch with at least one NaN value always return false. The implementation of Mahalanobis considers an inference as OOD if the confidence score is less than or equal to the threshold, and therefore it considers a confidence score with NaN as OOD. Still, such NaNs cannot be detected without any NaN checkings, which are embedded in my *ProGIP* implementation. In addition, *ProGIP* still effectively detects other NaN-indominant failures such as

ID to OOD failures in the backward pass of Mahalanobis and all types of failures in ODIN. I observed that excluding all of the NaN cases, *ProGIP* still can detect around 96.78% classification and ID/OOD detection failures.

Finally, I analyzed the false soft error detection rate of Ranger and *ProGIP* in the absence of faults. The detection-only Ranger encountered 0.0045% false soft error detection from originally correct executions in fault-free runs. On the other hand, *ProGIP* never encountered false soft error detection from originally correct executions in fault-free runs. *ProGIP* shows less false detection rate than the detection-only Ranger, since the lazy checking of *ProGIP* can expect some of the faults can be masked before the checking.

Chapter 6

CONCLUSION

As deep learning systems increasingly permeate safety-critical domains such as autonomous driving and disease diagnosis, the demand for reliable performance has become paramount. To address this urgent need, I present *ProGIP*, an advanced mechanism designed to detect soft errors in conjunction with confidence-based out-of-distribution (OOD) detection systems. By augmenting any existing confidence-based identification and OOD detection framework, *ProGIP* aims to enhance the overall reliability of deep learning applications significantly.

While *ProGIP* can function independently within a neural network, its true potential is realized through seamless integration with established gradient-based input perturbation OOD detection systems like ODIN and Mahalanobis. This integration ensures optimal efficiency and runtime performance. My method has been meticulously developed to identify critical soft errors, achieving impressive detection rates with minimal checkpoint insertions. Furthermore, *ProGIP* offers the flexibility to fine-tune soft error detection thresholds, allowing for tailored adjustments that meet the specific criticality requirements of various applications.

Importantly, My experiments demonstrate that implementing *ProGIP* incurs minimal runtime overhead, preserving the system's operational efficiency. I also highlight how variations in the choice of confidence function can influence the manifestation of errors, while the foundational methodology remains consistent.

Looking ahead, I envision a future where *ProGIP* can be integrated with other OOD detection techniques, promising a more comprehensive approach to ensuring the reliability of deep learning systems. In conclusion, *ProGIP* represents a crucial ad-

vancement in strengthening the reliability of deep learning technologies, safeguarding their performance across a wide array of safety-critical applications.

REFERENCES

[1] Bal, S., C. S. Mummidi, V. D. C. Ferreira, S. Srinivasan and S. Kundu, "A novel fault-tolerant architecture for tiled matrix multiplication", in "2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)", pp. 1–6 (IEEE, 2023).

[2] Chen, Z., G. Li and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction", in "2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)", pp. 1–13 (IEEE, 2021).

[3] Chen, Z., G. Li, K. Pattabiraman and N. DeBardeleben, "Binfi: An efficient fault injector for safety-critical machine learning systems", in "Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis", pp. 1–23 (2019).

[4] Corbière, C., "Robust deep learning for autonomous driving", arXiv preprint arXiv:2211.07772 (2022).

[5] Cui, P. and J. Wang, "Out-of-distribution (ood) detection based on deep learning: A review", Electronics **11**, 21, URL `https://www.mdpi.com/2079-9292/11/21/3500` (2022).

[6] DeVries, T. and G. W. Taylor, "Learning confidence for out-of-distribution detection in neural networks", arXiv preprint arXiv:1802.04865 (2018).

[7] Firouzi, F., M. E. Salehi, F. Wang and S. M. Fakhraie, "An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects", Microelectronics Reliability **51**, 2, 460–467 (2011).

[8] Gavarini, G., D. Stucchi, A. Ruospo, G. Boracchi and E. Sanchez, "Open-set recognition: an inexpensive strategy to increase dnn reliability", in "2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)", pp. 1–7 (IEEE, 2022).

[9] Ghavami, B., M. Sadati, Z. Fang and L. Shannon, "Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions", in "2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)", pp. 1239–1244 (IEEE, 2022).

[10] Goodfellow, I. J., J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples", in "International Conference on Learning Representations (ICLR)", (2015).

[11] Grigorescu, S., B. Trasnea, T. Cocias and G. Macesanu, "A survey of deep learning techniques for autonomous driving", Journal of field robotics **37**, 3, 362–386 (2020).

[12] Guo, C., G. Pleiss, Y. Sun and K. Q. Weinberger, "On calibration of modern neural networks", in "International conference on machine learning", pp. 1321–1330 (PMLR, 2017).

[13] Haque, S., A. W. Liu, S. Liu and J. H. Chan, "Improving the robustness of a convolutional neural network with out-of-distribution data fine-tuning and image preprocessing", in "Proceedings of the 12th International Conference on Advances in Information Technology", pp. 1–7 (2021).

[14] Hendrycks, D. and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks", in "International Conference on Learning Representations (ICLR)", (2017).

[15] Hinton, G., O. Vinyals and J. Dean, "Distilling the knowledge in a neural network", arXiv preprint arXiv:1503.02531 (2015).

[16] Hsu, Y.-C., Y. Shen, H. Jin and Z. Kira, "Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data", in "Proceedings of the IEEE/CVF conference on computer vision and pattern recognition", pp. 10951–10960 (2020).

[17] Huang, G., Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely connected convolutional networks", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 4700–4708 (2017).

[18] Jiang, W., Y. Ge, H. Cheng, M. Chen, S. Feng and C. Wang, "Read: Aggregating reconstruction error into out-of-distribution detection", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 37, pp. 14910–14918 (2023).

[19] Kononenko, I., "Machine learning for medical diagnosis: history, state of the art and perspective", Artificial Intelligence in medicine **23**, 1, 89–109 (2001).

[20] Lai, Z., K. Wei, Y. Fu, P. Härtel and F. Heide, "-prox: Differentiable proximal algorithm modeling for large-scale optimization", ACM Transactions on Graphics (TOG) **42**, 4, 1–19 (2023).

[21] Lee, K., URL `https://github.com/pokaxpoka/deep_Mahalanobis_detector/` (2018).

[22] Lee, K., K. Lee, H. Lee and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks", Advances in neural information processing systems **31** (2018).

[23] Li, G., S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications", in "Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis", pp. 1–12 (2017).

[24] Li, G., K. Pattabiraman and N. DeBardeleben, "Tensorfi: A configurable fault injector for tensorflow applications", in "2018 IEEE International symposium on software reliability engineering workshops (ISSREW)", (IEEE, 2018).

[25] Li, S., Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training", arXiv preprint arXiv:2006.15704 (2020).

[26] Li, Y., URL `https://github.com/facebookresearch/odin` (2018).

[27] Liang, S., Y. Li and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks", in "International Conference on Learning Representations (ICLR)", (2018).

[28] Liu, W., X. Wang, J. Owens and Y. Li, "Energy-based out-of-distribution detection", Advances in neural information processing systems **33**, 21464–21475 (2020).

[29] Mukherjee, S. S., J. Emer and S. K. Reinhardt, "The soft error problem: An architectural perspective", in "11th International Symposium on High-Performance Computer Architecture", pp. 243–247 (IEEE, 2005).

[30] Oh, H., J. Lee, H. Kim and J. Seo, "Out-of-order backprop: An effective scheduling technique for deep learning", in "Proceedings of the Seventeenth European Conference on Computer Systems", pp. 435–452 (2022).

[31] Ozen, E. and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications", in "2019 IEEE 28th Asian Test Symposium (ATS)", pp. 7–75 (IEEE, 2019).

[32] Pang, G., C. Shen, L. Cao and A. V. D. Hengel, "Deep learning for anomaly detection: A review", ACM computing surveys (CSUR) **54**, 2, 1–38 (2021).

[33] Ran, X., M. Xu, L. Mei, Q. Xu and Q. Liu, "Detecting out-of-distribution samples via variational auto-encoder with reliable uncertainty estimation", Neural Networks **145**, 199–208 (2022).

[34] Reis, G. A., J. Chang, N. Vachharajani, R. Rangan and D. I. August, "Swift: Software implemented fault tolerance", in "International symposium on Code generation and optimization", pp. 243–254 (IEEE, 2005).

[35] Ruff, L., J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich and K.-R. Müller, "A unifying review of deep and shallow anomaly detection", Proceedings of the IEEE **109**, 5, 756–795 (2021).

[36] Schorn, C. and L. Gauerhof, "Facer: A universal framework for detecting anomalous operation of deep neural networks", in "2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)", pp. 1–6 (IEEE, 2020).

[37] Schorn, C., A. Guntoro and G. Ascheid, "Efficient on-line error detection and mitigation for deep neural network accelerators", in "Computer Safety, Reliability, and Security: 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19-21, 2018, Proceedings 37", pp. 205–219 (Springer, 2018).

[38] Shafique, M., A. Marchisio, R. V. W. Putra and M. A. Hanif, "Towards energy-efficient and secure edge ai: A cross-layer framework iccad special session paper", in "2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)", pp. 1–9 (IEEE, 2021).

[39] Tian, Y., S. Ma, M. Wen, Y. Liu, S.-C. Cheung and X. Zhang, "To what extent do dnn-based image classification models make unreliable inferences?", Empirical Software Engineering **26**, 5, 84 (2021).

[40] Vernekar, S., A. Gaurav, V. Abdelzad, T. Denouden, R. Salay and K. Czarnecki, "Out-of-distribution detection in classifiers via generation", arXiv preprint arXiv:1910.04241 (2019).

[41] Vyas, A., N. Jammalamadaka, X. Zhu, D. Das, B. Kaul and T. L. Willke, "Out-of-distribution detection using an ensemble of self supervised leave-out classifiers", in "Proceedings of the European conference on computer vision (ECCV)", pp. 550–564 (2018).

[42] Wan, Z., A. Anwar, A. Mahmoud, T. Jia, Y.-S. Hsiao, V. J. Reddi and A. Raychowdhury, "Frl-fi: Transient fault analysis for federated reinforcement learning-based navigation systems", in "2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)", pp. 430–435 (IEEE, 2022).

[43] Xu, J., S. Zhu, Z. Li and C. Xu, "Joint distribution across representation space for out-of-distribution detection", arXiv preprint arXiv:2103.12344 (2021).

[44] Yang, D., K. Mai Ngoc, I. Shin, K.-H. Lee and M. Hwang, "Ensemble-based out-of-distribution detection", Electronics **10**, 5, 567 (2021).

[45] Yang, D., I. Shin, M. N. Kien, H. Kim, C. Yu and M. Hwang, "Out-of-distribution detection based on distance metric learning", in "The 9th International Conference on Smart Media and Applications", (2020).

[46] Yang, J., K. Zhou, Y. Li and Z. Liu, "Generalized out-of-distribution detection: A survey", arXiv preprint arXiv:2110.11334 (2021).

[47] Yang, J., K. Zhou, Y. Li and Z. Liu, "Generalized out-of-distribution detection: A survey", International Journal of Computer Vision pp. 1–28 (2024).

[48] Zhang, J., Q. Fu, X. Chen, L. Du, Z. Li, G. Wang, S. Han, D. Zhang *et al.*, "Out-of-distribution detection based on in-distribution data patterns memorization with modern hopfield energy", in "The Eleventh International Conference on Learning Representations", (2022).

[49] Zhang, J., J. Yang, P. Wang, H. Wang, Y. Lin, H. Zhang, Y. Sun, X. Du, K. Zhou, W. Zhang, Y. Li, Z. Liu, Y. Chen and H. Li, "Openood v1.5: Enhanced benchmark for out-of-distribution detection", arXiv preprint arXiv:2306.09301 (2023).

[50] Zisselman, E. and A. Tamar, "Deep residual flow for out of distribution detection", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 13994–14003 (2020).

APPENDIX A

ADDITIONAL RESULTS

Table A.1: Bit-wise fault injection results

	ODIN			Mahalanobis		
Faulty bit	NaN or inf.	Class. fail.	ID or OOD detect. fail.	NaN or inf.	Class. fail.	ID or OOD detect. fail.
31	886	6/**6**	14/**14**	10343	0/**0**	3/**3**
30	0	2072/**3**	2620/**9**	0	424/**0**	239/**3**
29	0	1/**0**	20/**3**	0	2/**0**	105/**3**
28	0	4/**0**	27/**3**	0	1/**0**	82/**1**
27	0	2/**1**	78/**25**	0	4/**0**	74/**3**
26	0	7/**7**	29/**12**	0	0/**0**	13/**13**
25	0	1/**1**	28/**23**	0	1/**1**	10/**9**
24	0	1/**1**	11/**11**	0	2/**2**	6/**6**
23	0	0/**0**	4/**4**	0	0/**0**	3/**3**
22	0	1/**1**	2/**2**	0	0/**0**	1/**1**
21	0	0/**0**	4/**4**	0	0/**0**	1/**1**
20∼0	0	0/**0**	0/**0**	0	0/**0**	0/**0**

Failure numbers N/**M** means (without ProGIP / **with ProGIP**)

Table A.1 shows the bit-wise number of NaNs, infinite numbers, and failures in ODIN and Mahalanobis, with and without *ProGIP*. Most of the NaNs and failures are due to the bit-flip at 30-bit, which is the highest exponent bit. The bit-wise results in Table A.1 represent that high-order bit-flips contribute most of the failures, and *ProGIP* can effectively detect most such faults. On the other side, Mahalanobis encountered 10,343 outputs with NaN or infinite value(s), whereas ODIN only encountered 886 such cases, due to the characteristics of the different scoring functions they utilize. This is why *ProGIP* can only detect 2 ID to OOD detection failures in ResNet with Mahalanobis; most of the detectable high deviations result in NaN outputs rather than failures in Mahalanobis. Note that we excluded NaNs that can be easily covered by *ProGIP* from the failure metrics, *ProGIP* still can detect around 97.0% failure-inducing soft errors. Note: These results are conducted on independent and separate experiments and hence they do not match the exact numbers in chapter 5. But, the trends of both sets of results are very similar.

APPENDIX B

OOD DETECTION WITH GIP

The taxonomy of OOD detection methods is vast and is beyond the scope of this thesis. Out-of-distribution detection can be done in several ways [46], the two broader approaches are supervised and semi-supervised [5]. The supervised approaches include different threshold-based [40], distance-based [6, 43] and density-based [28] approaches. The semi-supervised approach majorly includes the reconstruction error of the autoencoder to determine whether it is an ID sample or an OOD sample. Ran et al. [33] proposed an improved noise contrast prior (INCP) method to obtain reliable uncertainty estimates of standard VAE. No single method can consistently outperform others across benchmarks, and the ranking of their performance is different from one dataset to another [49]. An application should choose the OOD detection method based on its requirements.

The OOD detection with gradient-based perturbation requires three passes, a first forward pass and corresponding backward pass to calculate a gradient and another forward pass with perturbed input based on the gradient. Figure 2.1 is a high-level view of OOD detection techniques with GIP. As shown in Figure 2.1, the first forward pass performs classification with the sampled input. The backward pass is utilized for calculating the perturbation by backpropagating the gradient on the loss of the first forward pass. The perturbations of very low magnitude are then added to the same input. This perturbed input is again passed through the network which is the second forward pass of GIP. After the second forward pass a confidence scoring function is deployed to distinguish between ID and OOD inputs.

The choice of confidence scoring decides the effectiveness of the OOD detection solutions; among OOD detection solutions with GIP [27, 16, 22, 13, 45, 44, 50, 41, 48, 18], in this thesis, I focus on the confidence scoring of two representative OOD detections with gradient-based input perturbation, ODIN [27], and Mahalanobis [22]. ODIN is the first technique that utilized gradient-based input perturbation (GIP) to maximize the confidence score gaps between ID and OOD for OOD detection. Mahalanobis shares the GIP structure with ODIN but replaces the softmax-based confidence scoring of ODIN with Mahalanobis distance, which shows considerably different behaviors under the soft errors compared to the softmax. For the sake of simplicity, I exclude the feature ensemble approach of Mahalanobis to focus on the different effects and symptoms of the soft errors under different confidence scorings.

B.1 Input Perturbation

The primary goal of input preprocessing in the GIP approach is to add small perturbations to the input, which increases the confidence score of the inferences but tends to have a dominant effect on the ID inputs. Such input perturbations can maximize the confidence gap between ID and OOD inputs. Inspired by the fast gradient sign method [10]. The input perturbation is based on the Equation B.1.

$$\tilde{x} = x - \epsilon * sign(-\nabla_x score(x)) \tag{B.1}$$

In Equation B.1, x and $\tilde{x}$ represent the original and perturbed inputs, respectively. $(-\nabla_x score(x))$ is the gradient of the confidence function with respect to sample input. ODIN and Mahalanobis each use a different confidence scoring function.

According to the official implementation of both ODIN and Mahalanobis, the gradients are normalized to binary and then normalized to the same image space. When the gradients are normalized to binary instead of {0, 1}, they are converted to

{-1, 1}. The gradients are further normalized exactly as the image values. Further, the sign of these gradients is multiplied by the epsilon, and the resultant is added to the input image. The value of epsilon is so low that the authors claim and prove that this small perturbation effect on in-distribution is higher than the out-of-distribution images making them more separable. When the gradients are converted to -1, 1 the errors of the first forward pass lose their symptom which might lead to critical errors in the subsequent passes. Hence, I decided on a checkpoint before the gradients were normalized to binary. I will discuss this more in the subsequent sections.

B.2 ODIN

ODIN uses SoftMax with temperature scaling. Prior studies that utilized temperature scaling to distill the knowledge in a neural network [15] or to calibrate the confidence [12], ODIN applies the temperature scaling to the inputs of the SoftMax functions (logits) for better segregation between OOD and ID inputs. These scores are generated by:

$$Si(x;T) = \frac{exp(fi(x)/T)}{\sum_{j=1}^{N} exp(fj(x)/T)} \tag{B.2}$$

and are controlled by a hyperparameter T, where $fi(x)$ is the max logit for sample i without temperature scaling. The authors of ODIN observed that the segregation benefits from the temperature scaling are saturated after enough T value (T >100), and therefore they select $T = 1000$ for their implementation.

B.3 Mahalanobis

Mahalanobis is a distance-based technique and uses the Mahalanobis distance to calculate how far the perturbed input is from the mean of the class the input was classified as in the first forward pass. The confidence score based on Mahalanobis distance is calculated as follows:

$$M(x) = max_c \left\{ - \left((f(x) - \hat{\mu_c})^T \hat{\Sigma}^{-1} (f(x) - \hat{\mu_c}) \right) \right\} \tag{B.3}$$

The above equation is the Mahalanobis distance between the test sample x and the closest class-conditional Gaussian distribution where c is the index of the closest class, $\hat{\mu}_c$ is the sampled mean of the class c and $f(x)$ is pre-trained features of the SoftMax based neural classifier and $\hat{\Sigma}$ is the covariance matrix. The larger the distance further the sample from the mean of class. The input is classified as an OOD if the distance is greater than a particular threshold.

B.4 Deciding Threshold

The OOD detector of each of these methods consumes the confidence score from the second forward pass. The threshold equation can be described as:

$$g(x) = \begin{cases} OOD & if \quad score(\tilde{x}) \leq \delta \\ ID & if \quad score(\tilde{x}) > \delta \end{cases} \tag{B.4}$$

In Equation B.4, δ represents the threshold to classify ID/OOD, $g(x)$ represents the final confidence score of the respective methods, x and $\tilde{x}$ represent the original and perturbed inputs, respectively. If the confidence is higher than the threshold δ then the input is classified as ID else it is classified as OOD. A higher threshold increases the chances of correctly detecting OOD. Still, it can also increase the chances of the ID input with lower confidence scores being categorized as OOD. On the other hand, a lower threshold will decrease both the ratios of correctly detected OODs and incorrectly detected IDs. For the implementation of ODIN and Mahalanobis, I consider ID as positive and OOD as negative According to the metric of ODIN [27], and select the threshold at 95% true positive rate (TPR), i.e., the threshold that misidentifies 5% of IDs as OOD and correctly identifies 95% of IDs as ID.

ODIN and Mahalanobis both use input perturbation based on their respective confidence scoring functions, calculate the confidence of the perturbed input with their respective confidence scoring functions, and classify whether a sampled input is ID or OOD based on the decided threshold. I propose to detect soft errors in these techniques effectively and efficiently to make the models more reliable.