Check for updates

ACM DIGITAL LIBRARY | Association for Computing Machinery | acm open

PDF Download
3761796.pdf
13 January 2026
Total Citations: 0
Total Downloads: 470

Latest updates: https://dl.acm.org/doi/10.1145/3761796

Published: 03 October 2025
Online AM: 27 August 2025
Accepted: 12 August 2025
Revised: 11 August 2025
Received: 11 August 2025

Citation in BibTeX format

RESEARCH-ARTICLE

ProGIP: Protecting Gradient-based Input Perturbation Approaches for OOD Detection From Soft Errors

SUMEDH SHRIDHAR JOSHI, Arizona State University, Tempe, AZ, United States

HWISOO SO, Kyungpook National University (KNU), Daegu, South Korea

SOYEONG PARK, Yonsei University, Seoul, South Korea

WOOBIN KO, Yonsei University, Seoul, South Korea

JINHYO JUNG, Yonsei University, Seoul, South Korea

YOHAN KO, Yonsei University, Seoul, South Korea

View all

Open Access Support provided by:

Yonsei University

Arizona State University

Kyungpook National University (KNU)

School of Computing and Augmented Intelligence

Ewha Womans University

ProGIP: Protecting Gradient-Based Input Perturbation Approaches for OOD Detection from Soft Errors

SUMEDH SHRIDHAR JOSHI, Computer Science, Arizona State University, Tempe, United States
HWISOO SO, Computer Department, Kyungpook National University, Daegu, Korea (the Republic of), Department of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of), and School of Computing and Augmented Intelligence, Arizona State University, Tempe, United States
SOYEONG PARK, Departmet of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of)
WOOBIN KO, Department of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of)
JINHYO JUNG, Department of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of)
YOHAN KO, Departmet of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of)
UIWON HWANG, Department of Computer Science and Engineering, Ewha Womans University, Seodaemun-gu, Korea (the Republic of)

Sumedh Shridhar Joshi and Hwisoo So contributed equally to this research.

This work was partially supported by funding from National Science Foundation grants ACED 2436016 and Semiconductor Research Corporation (SRC) project 3154; by the MSIT (Ministry of Science and ICT), Korea, under the National Program in Medical AI Semiconductor) (2024-0-0097) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation) in 2025; by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2025-02304331,Digital Columbus Project); by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-00561169); by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)); by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2024-00341794); by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022-00165225); by the Korean Government (MSIT) (No. RS-2020-II201361 , Artificial Intelligence Graduate School Program (Yonsei University)); by the Regional Innovation System & Education (RISE) program through the Gangwon RISE Center, funded by the Ministry of Education (MOE) and the Gangwon State (G.S.), Republic of Korea. (2025-RISE-10-006).

Authors' Contact Information: Sumedh Shridhar Joshi, Computer Science, Arizona State University, Tempe, Arizona, United States; e-mail: sumedhjoshi1409@gmail.com; Hwisoo So, Computer Department, Kyungpook National University, Daegu, Korea (the Republic of) and Department of Computer Science, Yonsei University, Seodaemun-gu, Seoul, Korea (the Republic of) and School of Computing and Augmented Intelligence, Arizona State University, Tempe, Arizona, United States; e-mail: hwisoo.so@knu.ac.kr; Soyeong Park, Departmet of Computer Science, Yonsei University, Seodaemun-gu, Seoul, Korea (the Republic of); e-mail: psy980823@yonsei.ac.kr; Woobin Ko, Department of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of); e-mail: woobin.ko@yonsei.ac.kr; Jinhyo Jung, Department of Computer Science, Yonsei University, Seodaemun-gu, Seoul, Korea (the Republic of); e-mail: jinhyo.jung@yonsei.ac.kr; Yohan Ko (corresponding author), Departmet of Computer Science, Yonsei University, Seodaemun-gu, Seoul, Korea (the Republic of); e-mail: Yohan.Ko@yonsei.ac.kr; Uiwon Hwang, Department of Computer Science and Engineering, Ewha Womans University, Seodaemun-gu, Seoul, Korea (the Republic of); e-mail: uiwon.hwang@ewha.ac.kr; Kyoungwoo Lee, Department of Computer Science, Yonsei University, Seodaemun-gu, Seoul, Korea (the Republic of); e-mail: kyoungwoo.lee@yonsei.ac.kr; Aviral Shrivastava, School of Computing and Augmented Intelligence, Arizona State University, Tempe, Arizona, United States; e-mail: avirals@gmail.com.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2025 Copyright held by the owner/author(s).
ACM 1539-9087/2025/09-ART118
https://doi.org/10.1145/3761796

KYOUNGWOO LEE, Department of Computer Science, Yonsei University, Seodaemun-gu, Korea (the Republic of)
AVIRAL SHRIVASTAVA, School of Computing and Augmented Intelligence, Arizona State University, Tempe, United States

Undetected out-of-distribution (OOD) inputs pose a significant threat to the reliability of deep learning models, as they may lead to unexpected behaviors during inference. Several studies have proposed effective OOD input detection methods. However, soft errors—another significant threat to reliability—can impact both the classification results of neural network models and the ID/OOD detections of OOD detection methods. To provide a resilient OOD detection solution against soft errors, we analyze the effect of soft errors on neural network models with gradient-based input perturbation (GIP) approaches, which are representative methods for OOD detection. Building on our analysis, we propose ProGIP, which incorporates two software-level range-based fault detectors to protect all execution phases of GIP approaches, including two forward passes and one backward pass. Because it is purely software-based and adds just two scalar comparisons, ProGIP is readily deployable even on resource-constrained embedded platforms. Our ProGIP solution enables GIP approaches to distinguish between ID, OOD, and fault-affected inferences, detecting 97.7% of critical faults with a negligible runtime overhead of only 0.84%. Experimental results with 2.4 million fault injections across various neural networks and OOD detection methods demonstrate ProGIP's effectiveness in ensuring comprehensive reliability against non-malicious threats.

CCS Concepts: • **Hardware → Safety critical systems**; **Error detection and error correction**; • **Computing methodologies → Neural networks;**

Additional Key Words and Phrases: Soft error, transient fault, out-of-distribution (OOD), reliability, fault tolerance, neural network

ACM Reference Format:
Sumedh Shridhar Joshi, Hwisoo So, Soyeong Park, Woobin Ko, Jinhyo Jung, Yohan Ko, Uiwon Hwang, Kyoungwoo Lee, and Aviral Shrivastava. 2025. ProGIP: Protecting Gradient-Based Input Perturbation Approaches for OOD Detection from Soft Errors. *ACM Trans. Embedd. Comput. Syst.* 24, 5s, Article 118 (September 2025), 25 pages. https://doi.org/10.1145/3761796

1 Introduction

With the remarkable success of deep neural networks, machine learning has become pivotal in embedded systems. **Out-of-distribution** (**OOD**) data represents one major non-malicious threat to safety-critical machine learning systems. Since machine learning models are designed to learn from training datasets with specific distributions, they often lack the necessary information to process OOD data, which deviates significantly from **in-distribution** (**ID**) data.

Such OOD data may have various inputs with the same label as ID or completely different labels from ID [52]. Inference with the former case can significantly decrease the model's classification accuracy. Furthermore, correct inference in the latter case is often impossible, as the original model provides no information for the unknown labels.

Therefore, to ensure the reliability of artificial intelligence systems, it is essential to detect OOD data and reject such unfamiliar cases before the machine learning system behaves abnormally by producing unreliable inference results with OOD data [46]. For instance, if an autonomous driving system encounters objects that were not encountered during training, it should recognize the anomaly and issue a warning [5, 52].

Among various OOD detection approaches, **Gradient-based Input Perturbation** (**GIP**) is theoretically appealing due to its conceptual simplicity and effectiveness. By introducing small

perturbations derived from input gradients, GIP explicitly maximizes the model's confidence disparity between ID and OOD inputs, thereby improving discriminative performance for OOD detection. GIP-based methods—most notably ODIN [32] and Mahalanobis [27]—are now used as standard baselines in recent public OOD benchmark suites, such as OpenOOD [54] and G-OSR [9].

1.1 Why Protecting GIP Solutions is Important?

GIP, although effective at OOD detection, is itself vulnerable to another reliability threat: soft errors. A soft error is a transient fault—a single-event bit flip in a transistor—typically induced by alpha particles, thermal neutrons, or cosmic rays. [34]. Previous research has observed that a single bit-flip can alter the classification result of deep learning models [28].

Our experiments also show that soft errors in neural networks can induce not only incorrect classification results but also erroneous ID/OOD decisions of GIP solutions. In neural networks, these bit-flips on execution units can alter neuron outputs. The architectures of GIP solutions involve multiple passes (forward, backward, and forward), which we will explain in detail in the upcoming sections. Even a single bit-flip can severely distort gradients and confidence scores compared to standard inference.

GIP methods require a precise computation of gradients and confidence scores. Soft errors can severely affect their effectiveness by introducing miscalculations. We define two types of critical failure that can occur in neural networks with GIP-based OOD detection due to soft errors.

- **Classification failures:** When a model correctly classifies an input in the absence of soft errors but misclassifies it due to a soft error.
- **ID/OOD detection failures:** When a model correctly distinguishes between ID and OOD inputs in the absence of soft errors but incorrectly classifies ID as OOD or vice versa due to a soft error.

Soft errors are highly unpredictable and can occur randomly at any time, potentially causing catastrophic misjudgments in intelligent, real-time systems. The existing OOD detection systems do not consider soft errors. This represents a significant gap in achieving reliability in intelligent safety-critical systems.

Existing solutions do not provide comprehensive reliability against both OOD data and soft errors. Few solutions can detect both threats as outliers, but they cannot distinguish whether the detected outlier is a case of OOD data or a soft error.

Distinguishing between OOD data and soft errors is crucial for developing targeted countermeasures against specific threats. For example, operators of machine learning applications can collect detected OOD inputs as potential learning resources for future use [38, 40, 52]. On the other hand, since voltage and frequency affect the soft error rate [10], system designers can monitor soft error occurrences to adjust **dynamic voltage and frequency scaling** (**DVFS**) settings [44]. Such threat-specific actions are impossible without a method to distinguish 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 [15, 20, 22, 27, 32, 48, 51, 53] require one forward and one backward pass 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, we propose **ProGIP** (**Protecting Gradient-based Input**

Perturbation), which protects GIP approaches for OOD detection against soft errors with minimal software-level detectors.

Our approach is based on range-based checkers [3, 12] that can distinguish between ID, OOD, and fault-affected cases. Inspired by the observation in previous studies [3, 12] that faults in high-order bits of the model primarily contribute to failures, this article analyzes the effects and symptoms of high-order bit-flips in ID/OOD detection results, along with classification results.

While we primarily focus on high-order bit-flips due to their significant impact and detectability, we acknowledge that lower and middle-order bit-flips can also cause marginal classification errors. Based on our analysis, we develop 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 is placed immediately after the backward pass to cover the first forward and backward passes, and another is placed after the second forward pass to cover that pass. The main contributions of this article are as follows:

— We analyze the impact of soft errors on neural networks using GIP approaches, identifying vulnerabilities in existing OOD detection methods.
— We propose ProGIP, a holistic reliability solution that integrates only two fault detectors to cover all three (forward, backward, forward) passes in GIP approaches, effectively distinguishing between OOD data and soft errors.
— Our method detects 97.7% of critical failures in neural network models using the GIP solution, with only 0.84% runtime overhead, ensuring comprehensive reliability for deep learning models against non-malicious threats.
— Our approach provides a versatile protection framework, where a minimal core of just two range checks ensures scalability from GPUs to microcontrollers. The optional addition of a single checker before the activation-modification stage effectively hardens a broader class of state-of-the-art OOD detectors.

2 Background

2.1 Gradient-Based Input Perturbation (GIP) Solutions for Out-of-Distribution (OOD) Detection

OOD detection can be approached in several ways [52], with the two broader categories being supervised and semi-supervised [49]. The supervised approaches include different threshold-based [47], distance-based [25, 52], and density-based [33] methods. The semi-supervised approach primarily includes reconstruction error measurements from autoencoders to determine whether a sample is ID or OOD. Ran et al. [28] proposed an **improved noise contrast prior** (**INCP**) method to obtain reliable uncertainty estimates using standard VAE.

A fundamental strategy for distinguishing ID and OOD data is to score the confidence of inference results based on softmax values, building on the observation that the highest softmax values of ID inferences tend to be higher than those of OOD inferences [18]. No existing single method can consistently outperform others across all benchmarks, and their performance ranking varies from one dataset to another [54].

In this article, we focus on the GIP approach, one of the most representative methods in OOD detection. For simplicity, instead of introducing all GIP solutions [15, 20, 22, 27, 32, 48, 51, 53] that share similar structures, this article primarily discusses two representative GIP solutions: ODIN [32] and Mahalanobis [27].

2.2 GIP Approach Overview

The primary objective of the GIP approach is to maximize the confidence score gaps between ID and OOD data by perturbing the input based on the gradient, thereby enabling GIP solutions

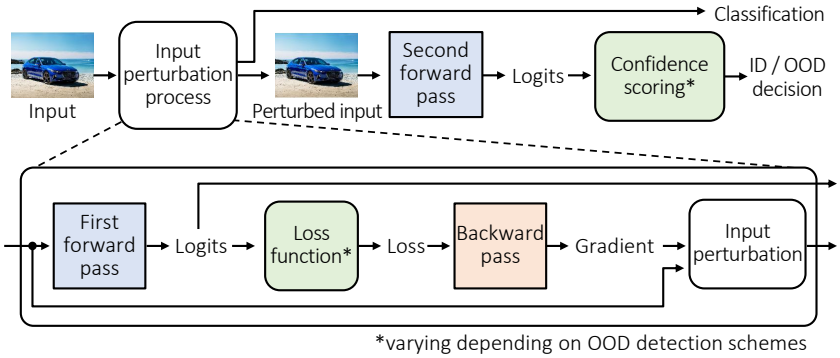


Fig. 1. GIP solutions process first forward and backward passes to perturb the input and second forward pass to distinguish ID and OOD.

to distinguish ID and OOD data more effectively. Figure 1 shows the high-level view of GIP solutions.

In Figure 1, the GIP approach first generates the perturbed input through an input perturbation process, which includes first forward and backward passes to generate the gradients. Inspired by the fast gradient sign method [13], the input perturbation in this process maximizes the confidence gap between ID and OOD inputs using Equation (1).

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

In Equation (1), x and $\tilde{x}$ represent the original and perturbed inputs, respectively. $(-\nabla_x score(x))$ represents the gradient of the scoring function concerning the sample input, where the scoring function varies based on solutions. The *sign* function preserves only 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 GIP is well-discussed i n the ODIN paper [32].

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.

2.3 Confidence Scoring Functions

The confidence scoring function in the GIP approach varies depending on the specific solution. ODIN [32] uses the maximum softmax value with temperature-scaled logits, which is utilized to distill knowledge in a neural network [19] or to calibrate confidence [14]. On the other hand, Mahalanobis [27] calculates the Mahalanobis distance between the logits generated with perturbed input and the mean of logits generated with ID inputs for each class, using the negative of the maximum Mahalanobis distance value as a confidence score. The higher the distance further the input is from the mean of the ID input class. In both solutions, the confidence scores of ID inputs tend to be higher than those of OOD inputs. Therefore, both solutions distinguish between ID and OOD by applying Equation (2):

$$g(x) = \begin{cases} OOD & if \quad score(\tilde{x}) \leq \delta \\ ID & if \quad score(\tilde{x}) > \delta \end{cases} \tag{2}$$

2.4 Threshold Selection and Classification Results

In Equation (2), δ represents the threshold to classify ID/OOD, $g(x)$ represents the final confidence score of the respective methods, and x and $\tilde{x}$ represent the original and perturbed inputs, respectively.

Table 1. Taxonomy of Failures

Category	Failure type	Description	Target of ProGIP
Silent data corruption (SDC, system-invisible failures)	Classification failure	The model would correctly predict the class of an in-distribution (ID) input when fault-free, but a transient bit-flip causes it to misclassify.	Yes
	ID/OOD detection failure (ID to OOD and OOD to ID)	The model would correctly distinguish an input between ID vs. OOD input when fault-free, but a bit-flip affects ID/OOD detection—either marking ID as OOD (ID to OOD) or vice versa (OOD to ID).	Yes
System-visible failures	Crash	The machine learning model execution has crashed (e.g., segmentation fault) due to a bit-flip	No
	Hang (timeout)	The machine learning model execution exceeds the expected execution time (e.g., infinite loop) due to a bit-flip	No

If the confidence is higher than the threshold δ, then the input is classified as ID; otherwise, it is classified as OOD. A higher threshold increases the chances of correctly detecting OOD input, but also increases the likelihood of incorrectly classifying ID inputs with lower confidence scores as OOD. For the implementation of ODIN and Mahalanobis, we consider ID as positive and OOD as negative according to the metric of ODIN [32] 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. It is important to note that the GIP solutions in Figure 1 do not utilize the classification results of the second forward pass with perturbed input, as the input perturbation may alter the classification results. Instead, they utilize the result of the first forward pass with the original input as the classification result.

3 Proposed Method: ProGIP

This section proposes ProGIP to combat **silent data corruption** (**SDC**), one of the most critical threats to the reliability of machine learning models. An SDC is a system-invisible failure: the model yields an erroneous result without any discernible symptoms (e.g., a crash or hang). Because the system remains unaware of the fault, it cannot initiate recovery actions. This stands in stark contrast to system-visible failures, which can be mitigated by strategies such as re-execution. ProGIP is a lightweight mechanism explicitly designed to detect the critical faults within GIP solutions that lead to SDCs. The complete failure taxonomy is presented in Table 1.

ProGIP aims to prevent two types of failures: (i) classification failures and (ii) ID/OOD detection failures. We define these failures as follows: **Classification failure**: A neural network model with an existing GIP solution can correctly classify the input in the absence of soft errors, but a soft error causes the model to misclassify the input. **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). We term any bit-flip that induces one or both of these failures a "critical fault" in this work. Several studies have demonstrated that the majority of soft-error-induced failures in neural networks are caused by high-order bit-flips [3, 12, 28].

3.1 Threat Analysis and Design Principles

Motivated by these observations, we analyze how high-order bit-flips on the three execution passes of GIP solutions induce these failures and what visible symptoms these bit-flips produce. Based on this analysis, ProGIP strategically places two fault detectors that can protect all three execution passes of GIP solutions with minimal overhead. While we primarily focus on high-order bit-flips

due to their significant impact on neural network outputs and their relatively straightforward detectability, we acknowledge that lower and middle-order bit-flips can also cause errors that might be harder to detect. Section 5 discusses this further and provides insights into the effectiveness of ProGIP against various types of bit-flips. The first fault detector (Section 3.2) detects abnormal gradient values to protect the first forward and backward passes. The second fault detector (Section 3.3) detects abnormal confidence scores to protect the second forward pass.

ProGIP is a purely software-based approach without internal modification of target neural networks. The two fault detectors of ProGIP only utilize the outputs of the passes in GIP approaches (gradient from the backward pass and OOD score from the second forward pass). Therefore, ProGIP does not require modification of the inference engine and can operate outside of the execution of the inference.

The goal of ProGIP is to provide lightweight fault detection. Once ProGIP detects a fault, the system offers several fallback options depending on the application's safety requirements: for example, the system can re-execute the exact inference pass to confirm the error (re-execution) or trigger a fail-safe mode in safety-critical scenarios (e.g., autonomous driving). For re-execution, if the system detects a fault at the second fault detector of ProGIP, it only needs to re-execute the second forward pass of the GIP approach rather than fully re-executing from the first forward pass. We leave detailed system-level responses after the detection for future work.

3.2 First Fault Detector: Gradient Checker for the First Forward and Backward Passes

3.2.1 Analysis of Effects of High-Order Bit-Flips. Figure 2 illustrates the effects of high-order bit-flips on the first forward and backward passes of GIP solutions. 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 can also 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 used 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:

$$max_c\{-(f(x)-\hat{\mu}_c)^T\hat{\Sigma}_c^{-1}(f(x)-\hat{\mu}_c)\} \tag{3}$$

In Equation (3), as the logits $f(x)$ increase, the confidence scores decrease since the confidence is the negative of the 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, 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 backward pass do not affect classification results.

3.2.2 Designing Fault Detector. 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 (1) only utilizes the sign of the gradient and disregards the magnitude. Note that

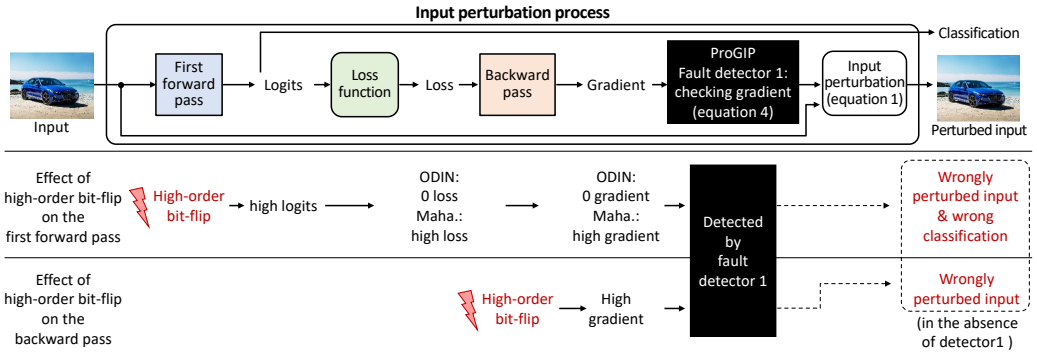


Fig. 2. High-order bit-flips on the first forward and backward passes in GIP solutions induce extremely high or near zero gradients, which is tricky to observe in error-free inferences. The first fault detector of ProGIP checks the gradient based on this observation to cover high-order bit-flips on such passes.

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, ProGIP places the first fault detector right before the input perturbation, which can be implemented at the software level with Equation (4).

$$check(grad) = \begin{cases} Faulty & if \quad max(abs(grad)) \approx 0, \\ Faulty & if \quad max(abs(grad)) > \delta_{F1}, \\ Fine & otherwise. \end{cases} \tag{4}$$

The first fault detector operates during the backward pass and requires minimal computational resources, as it performs only a simple comparison between the maximum gradient magnitude and predefined thresholds. This design choice ensures that ProGIP adds negligible overhead to the overall execution time.

3.2.3 Threshold Selection. In Equation (4), $grad$ 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(4), we 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}, we profile the $max(abs(grad))$ values from the inferences with the ID training dataset and find the maximum value of $max(abs(grad))$. Since naturally large gradients from the fault-free inference with a test dataset can cause false fault detection alarms, we conservatively add a 200% margin to the selected maximum value, i.e., triple it.

3.3 Second Fault Detector: OOD Score Checker for the Second Forward Pass

3.3.1 Analysis of Effects of High-Order Bit-Flips. Figure 3 illustrates the effects of high-order bit-flips on the second forward pass of GIP solutions. 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 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

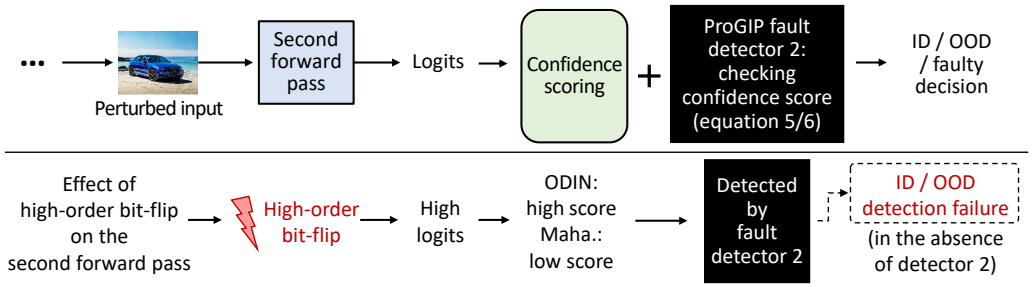


Fig. 3. High-order bit-flips on the second forward pass of GIP solutions induce extremely high or low confidence scores compared to fault-free inferences. Based on this observation, the second fault detector of ProGIP checks the confidence score to detect high-order bit-flips on the second forward pass.

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) will classify the input as ID, resulting in an ID/OOD detection failure (OOD to ID). For example, in ResNet with ODIN, we 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 will 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, we 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.

3.3.2 Designing Fault Detector. 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, 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 (5) shows how the second fault detector with the scoring function of ODIN distinguishes between ID, OOD, and faulty inferences.

$$g'_O(x) = \begin{cases} OOD & if \quad score_O(\tilde{x}) \leq \delta_{OOD}, \\ ID & if \quad \delta_{OOD} < score_O(\tilde{x}) \leq \delta_{F2}, \\ Faulty & if \quad score_O(\tilde{x}) > \delta_{F2}. \end{cases} \tag{5}$$

In Equation (5), 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 (6) shows how the second fault detector with the scoring function of Mahalanobis distinguishes between ID, OOD, and faulty inferences.

$$g'_M(x) = \begin{cases} Fault & if \quad score_M(\tilde{x}) \leq \delta_{F2}, \\ OOD & if \quad \delta_{F2} < score_M(\tilde{x}) \leq \delta_{OOD}, \\ ID & if \quad \delta_{OOD} < score_M(\tilde{x}). \end{cases} \tag{6}$$

In Equation(6), 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 lower than the threshold δ_{F2}.

Similar to the first fault detector, the second fault detector requires minimal computational resources as it is implemented within the existing ID/OOD decision logic, requiring only an additional threshold comparison. This efficient design ensures that ProGIP maintains a low overhead while providing comprehensive protection against soft errors.

3.3.3 Threshold Selection. As discussed in Section 2.1, we select ID/OOD threshold δ_{OOD} at 95% TPR for ID/OOD classification. To select the fault detection threshold δ_{F2}, we 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 Section 3.2. For δ_{F2} of ODIN, we 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, we find the minimum score during the training phase and triple it.

3.4 Handling Not a Number (NaN) and Infinity Values

In addition to the range-based checks described in Sections 3.2 and 3.3, ProGIP also incorporates checks for **not a number** (**NaN**) and infinity values. These special floating-point values often occur when high-order bit-flips lead to operations such as division by zero or the square root of a negative number. NaN values are particularly problematic because they propagate through computations: any operation involving a NaN results in another NaN. Infinity values, while mathematically defined, can also lead to unexpected behaviors in neural networks. Both NaN and infinity values can significantly impact the classification and ID/OOD detection results. To address this issue, ProGIP includes NaN and infinity checks at both fault detection points. This ensures that computations producing these special values are identified as faulty, preventing incorrect classification or ID/OOD detection decisions. These checks add minimal overhead to the fault detection process while significantly enhancing the fault coverage of ProGIP.

4 Experimental Setup

We conducted comprehensive fault injection and runtime measurement experiments to evaluate the efficiency and fault coverage of ProGIP. This section details our experimental methodology, including the networks and datasets used, the OOD detection methods implemented, the soft error detection techniques compared, and the fault injection and runtime measurement procedures.

Table 2. Network and OOD Detection Information

Network	ID Dataset	OOD detection	ID Classification accuracy	AUROC for TPR vs FPR curve	FPR at 95% TPR
DenseNet-BC [21]	CIFAR-10 [23]	ODIN [32]	95.19%	98.52%	7.46%
		Mahalanobis [27]		96.28%	15.36%
ResNet-34 [55]		ODIN [32]	94.61%	90.62%	38.25%
		Mahalanobis [27]		93.35%	33.53%
MobileNetV2 [41]		ODIN [32]	94.05%	93.50%	24.24%
		Mahalanobis [27]		90.44%	52.80%

4.1 Network, Dataset, and OOD Detection Methods

We adopted DenseNet-BC [21], ResNet-34 [55], and MobileNetV2 [41] as our neural network architectures,[1] trained to classify the CIFAR-10 [23] dataset. CIFAR-10 consists of 60k 32×32 color images in 10 classes, with 50k training images and 10k test images. For the OOD dataset, we used the Tiny ImageNet [6] test dataset consisting of 10k images resized to 32×32. For the DenseNet-BC implementation, we used the pre-trained network provided by the official implementation of ODIN [31], which was trained with a growth rate of 12 for 300 epochs with a weight decay of 10^{-4}. For ResNet34, we trained the network for 200 epochs with a weight decay of $5x10^{-4}$. For MobileNetV2, we used the implementation in an open-source repository [24] that adapted MobileNetV2 for the CIFAR-10 dataset and trained the MobileNetV2 for 200 epochs with a weight decay of $5x10^{-4}$. All models used a **Stochastic Gradient Descent** (**SGD**) optimizer and a learning rate of 0.1 (ResNet34, DenseNet-BC) or 0.01 (MobileNetV2), which was divided by 10 at 50% and 75% of the total number of training epochs. Table 2 summarizes the ID classification accuracy and OOD detection capability of each network and OOD detection scheme.

4.2 Soft Error Detection Methods

We implemented two software-level ProGIP checkers, as described in Sections 3.2 and 3.3, into the ODIN and Mahalanobis implementations. Further, we included checks for NaN and infinity values as these exceptional cases have unexpected behaviors in PyTorch's comparison operations.

To our knowledge, there is no existing solution specifically designed to provide soft error detection for GIP solutions. To establish a baseline for comparison, we implemented a detection-only version of Ranger [3], a state-of-the-art soft error detection method for neural networks. This detection-only Ranger checks for abnormal values at the outputs of all activation function layers as well as the last layer of the forward and backward passes using PyTorch's built-in hook methods. The original Ranger approach also attempts to correct faults by replacing abnormal values with predetermined safe values. However, to ensure a fair comparison with ProGIP, which only provides fault detection, our detection-only Ranger implementation only raises the alarm when abnormal values are detected without attempting correction. The NaN and infinity value checking is done by a checker shared by Ranger and ProGIP at the end of the backward pass and the second forward pass. Note that

[1]We intentionally chose three canonical CNN families that span the design space most often discussed in the embedded–systems literature. DenseNet-BC, in its 100-layer, k=12 variant (0.8M parameters), is characterized by a dense connectivity pattern that is extremely parameter-efficient but induces high feature-reuse traffic. ResNet-34 (21.8M parameters, 3.6 GFLOPs) is the de-facto medium-scale residual baseline [16] appearing in numerous software- and hardware-optimization studies [2, 50]. MobileNetV2 (3.5M parameters, 0.3 GFLOPs) represents the edge/IoT class. It couples depth-wise separable convolutions with inverted bottlenecks to minimize computing and memory and is widely deployed on mobile SoCs. Therefore, using these three networks lets us stress-test ProGIP across (i) densely connected, (ii) residual, and (iii) lightweight architectures with different compute–memory footprints.

this detection-only Ranger shares the same fault detection methodology as the original Ranger with fault correction. The only difference between the original Ranger and the detection-only Ranger is how to deal with the detected fault, correcting the fault with the boundary value in the original Ranger and just raising the fault detection alarm in the detection-only Ranger. Still, we acknowledge that the fine-grained fault detection of Ranger is to prevent fault propagation from one faulty neuron to other neurons.

To select the threshold values for both ProGIP and detection-only Ranger, we profiled the ID training dataset and found the maximum values in fault-free inferences. As discussed in Sections 3.1 and 3.2, we profiled two values for ProGIP and selected them as thresholds with a 200% margin. For the detection-only Ranger, we profiled the maximum and minimum values of the layers with the fault detector and added a 300% margin to utilize them as thresholds. The 200% and 300% margins were selected by increasing the margin by 50% until there was no false fault detection in fault-free inferences with ID and OOD test datasets.

4.3 Fault Injection Setup

We designed our fault injection experiments to simulate soft errors in the datapath of neural networks. Following the approach of several prior studies [3, 4, 28, 29, 43], we assumed that existing **Error Correction Code** (**ECC**) or parity mechanisms could effectively cover faults in memory [39], and therefore focused on faults occurring during computation.

To simulate bit-flips in the datapath, we implemented a fault injector that directly flips a random bit in one of the outputs of a randomly selected layer via PyTorch's hook methods. The rationale behind affecting only one output is that a transient fault on a logic component in the datapath for neural networks such as multiply-accumulate (MAC) units affects one output [17], while a fault on a memory component, which can be covered by ECC or parity, can affect multiple outputs. This approach allows us to mimic the effects of soft errors on the computation pipeline without modifying the underlying hardware. All fault injections other than runtime measurements were conducted on NVIDIA RTX 4070 and NVIDIA RTX A6000.

For each fault injection trial, we followed this procedure:

- Randomly select a layer in the execution pass (first forward, backward, or second forward).
- Execute an inference of the GIP solution with both ProGIP and the detection-only Ranger.
- Just before the execution of the selected pass, add a fault injection hook that will flip a random bit in the output of the selected layer.
- Execute the selected pass with the fault injection hook active.
- Remove the hook and continue the remaining execution.
- Collect classification and ID/OOD detection results, along with soft error detection results from both protection methods.

To identify classification and ID/OOD detection failures, we also executed the inference with the same input and random seeds without injecting faults and then compared the results between fault injection and fault-free runs.

We executed 100,000 fault injection trials per configuration, considering the OOD detection schemes (ODIN or Mahalanobis), data types (ID or OOD), network types (DenseNet-BC, ResNet-34, and MobileNetV2), and the passes affected by the fault (first forward, backward, or second forward pass)

Overall, we executed a total of 3.6M fault injection runs—100,000 injections × 2 (OOD detection schemes) × 2 (data types) × 3 (passes) × 3 (networks). Note that we injected one fault for each execution of an inference with ODIN or Mahalanobis. Thus, our fault injection experiments aim to estimate the failure probabilities of GIP approaches without or with the protection schemes if a fault occurs during the inference.

Table 3. Runtime Measurement Results

Network	OOD detection	Normalized execution time	
		Detection-only Ranger	ProGIP
DenseNet-BC [21]	ODIN [32]	256.58%	100.93%
	Mahalanobis [27]	259.01%	100.68%
ResNet-34 [55]	ODIN [32]	240.32%	101.07%
	Mahalanobis [27]	234.17%	100.56%
MobileNetV2 [41]	ODIN [32]	214.22%	100.97%
	Mahalanobis [27]	206.15%	100.85%
Average		235.07%	100.84%

This comprehensive fault injection campaign allowed us to thoroughly evaluate the effectiveness of ProGIP across various scenarios and compare it with the detection-only Ranger approach.

4.4 Runtime Measurement

To assess the efficiency of ProGIP, we measured the execution times of ODIN and Mahalanobis with three configurations:

— No soft error protection
— With the detection-only Ranger
— With ProGIP

We 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 to ensure a realistic and consistent execution environment. Each runtime measurement iteration consisted of executing inferences on 1,000 ID inputs and 1,000 OOD inputs without batching. We repeated this process for 20 iterations for each configuration.

To ensure robust results, we computed the mean execution time for each configuration, excluding the top and bottom 10% of iterations (to remove outliers). The results are normalized to the execution time of the unprotected baseline to quantify the overhead introduced by each protection mechanism. This experimental setup allowed us to thoroughly evaluate both the effectiveness and efficiency of ProGIP, providing comprehensive insights into its performance compared to existing approaches. The results of these experiments are presented and analyzed in the following section.

5 Experimental Results

In this section, we present and analyze the results of our experiments, focusing on two key aspects: runtime overhead and fault coverage. We first examine the efficiency of ProGIP by comparing its runtime overhead with that of the detection-only Ranger. Then, we analyze the fault coverage of ProGIP, detailing its effectiveness in detecting classification and ID/OOD detection failures across different execution passes, neural networks, and OOD detection methods.

5.1 Runtime Overhead Analysis

Table 3 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. These results demonstrate that ProGIP achieves significantly lower runtime overhead compared to the detection-only Ranger.

Table 4. Summary of the Fault Injection Experiments with CIFAR-10 as ID Dataset

Method	Fault injection run	Originally correct run*	Classification failure	ID/OOD detection failure
Unprotected (Original)	3,600,000	2,922,540	7,673	10,217
Detection-only Ranger			20	273
ProGIP (Ours)			33	379

* The execution that can produce correct classification and ID/OOD results if no fault is injected.

The detection-only Ranger induces an average runtime overhead of 135.07% across all configurations. This substantial overhead is primarily due to the backward hook methods applied to every ReLU layer, which significantly slows down the execution. Note that it is usual to customize the backward pass via hooks in PyTorch [25, 30, 35].

Delicately optimized custom kernel can reduce the runtime overhead of Ranger implementation. Still, Ranger should check values from all ReLU layers, while ProGIP only needs to check values from two detection points. In contrast to Ranger, ProGIP introduces only 0.84% average runtime overhead on average. This minimal overhead is achieved by strategically placing just two checkers at key points in the execution flow rather than monitoring every layer. The first checker is placed after the backward pass, and the second is integrated with the existing ID/OOD detection logic, requiring only an additional threshold comparison.

The low overhead of ProGIP makes it particularly suitable for real-time applications where execution speed is critical, such as autonomous driving systems. Further, the low overhead of ProGIP demonstrates that ProGIP is suitable for protecting machine learning models in embedded systems, where the computing resource is strictly constrained. By adding less than 1% to the execution time, ProGIP provides an efficient solution to improve the reliability of neural networks with GIP-based OOD detection.

5.2 Detailed Analysis of Fault Injection Experiments

Table 4 summarizes the fault injection results. In this table, the originally correct run represents the executions that produce correct classification (for ID inputs) and ID/OOD detection results if no fault occurs. Without protection, 0.263% and 0.350% of the faults resulted in classification and ID/OOD detection failures for the originally correct runs, respectively. The low failure probabilities upon a fault emphasize the necessity of efficient fault detection solutions with minimal runtime overhead. Detection-only Ranger detected 98.36% of total failures by detecting 99.74% classification failures and 97.33% ID/OOD detection failures. ProGIP detected 97.70% of total failures by detecting 99.57% classification failures and 96.29% ID/OOD detection failures. Overall, ProGIP shows comparable fault detection capability compared to the detection-only Ranger, while ProGIP induces 0.84% runtime overhead on average, as discussed in Section 5.1.

Figure 4 provides a detailed breakdown of the fault injection results, showing the number of failures for different types (classification, ID→OOD, OOD→ID), different execution passes (first forward, backward, second forward), and different configurations (unprotected, detection-only Ranger, ProGIP). In Figure 4, white bars represent the number of failures in the unprotected version, gray bars represent the detection-only Ranger version, and black bars represent the ProGIP version. A closer examination of the results for specific networks and OOD detection methods reveals additional insights, as shown in Figure 4. Several key observations are as follows:

— High coverage for critical faults: Both ProGIP and the detection-only Ranger effectively detect the vast majority of faults that cause classification or ID/OOD detection failures. This is particularly

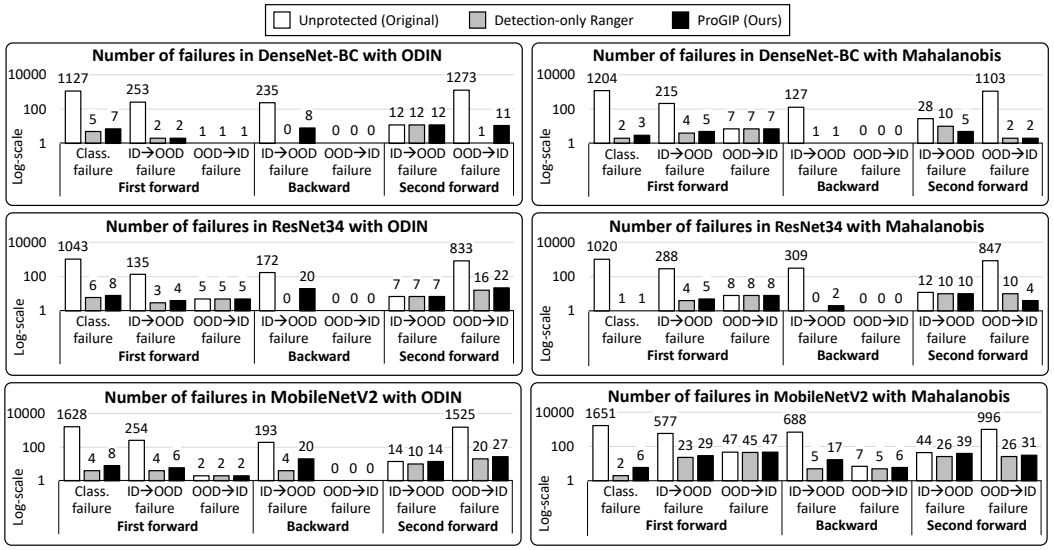


Fig. 4. The detailed fault injection results show that the coverage of ProGIP is comparable to the one of the detection-only Ranger, and ProGIP even outperforms the detection-only Ranger for a few cases (ID to OOD in DenseNet-BC with Mahalanobis and OOD to ID in ResNet34 with Mahalanobis).

evident for high-order bit-flips, which typically result in extreme output values. Extremely large values resulting from high-order bit-flips.

— OOD to ID failures in the second forward pass: In the second forward pass, OOD to ID failures are dominant compared to the ID to OOD failures in both ODIN and Mahalanobis. For ODIN, this is because abnormally high values resulted in high confidence scores. On the other hand, in Mahalanobis, most OOD to ID failures are due to the NaN values. Mahalanobis scoring with abnormally high values usually resulted in NaN scores, which will be discussed in Section 5.3.

— Limitations of the range-based solutions: Both detection-only Ranger and ProGIP cannot detect most of the ID to OOD failures in the second forward pass of ODIN. This is because such failures are due to the decreased confidence scores, while both solutions are designed to detect abnormally high values. Similarly, both solutions are hard to detect OOD to ID failures in the second forward pass of Mahalanobis except for such failures involving NaN scores; still, most of the OOD to ID failures in the second forward pass of Mahalanobis in Figure 2 produced NaN scores, and therefore both detection-only Ranger and ProGIP could detect them.

— A secondary effect of fault detection on the scores: While detection-only Ranger more frequently checks the faults compared to ProGIP, ProGIP shows better coverage for a few cases in Figure 4. At first, ProGIP outperforms detection-only Ranger for the ID to OOD failures in the second forward pass of DenseNet-BC with Mahalanobis. This is because we give a 200% margin for the fault thresholds of ProGIP, while we give a 300% margin for the ones of detection-only Ranger. Note that such different margins are selected by finding margin values that never induce false fault detection; selecting the proper margin for each layer can eliminate such gap, but it is hard to find such proper layer-wise margin in the design phase of the real-world environment that is hard to obtain OOD test set. In addition, ProGIP also outperforms detection-only Ranger for the OOD to ID failures in the second forward pass of ResNet34 with Mahalanobis. We observed that some failure-inducing faulty activation values do not show abnormally high maximum

Table 5. Bit-Wise Fault Injection Results for ODIN with and without ProGIP

Faulty bit	ODIN									
	Classification failure					ID/OOD detection failure				
	Number of failures	Detected by ProGIP based on				Number of failures	Detected by ProGIP based on			
		NaN check	Inf check	δ_{F1} check	Undetected		NaN check	Inf check	δ_{F1}, δ_{F2} check	Undetected
31 (sign bit)	4	0	0	0	4	10	0	0	0	10
30	3774	198	0	3567	9	4630	237	0	4367	26
29	2	0	0	2	0	19	0	0	18	1
28	8	0	0	8	0	29	0	0	22	7
27	2	0	0	0	2	130	0	0	93	37
26	2	0	0	0	2	38	0	0	13	25
25	3	0	0	0	3	24	0	0	3	21
24	3	0	0	0	3	18	0	0	0	18
23	0	0	0	0	0	9	0	0	0	9
22	0	0	0	0	0	3	0	0	0	3
21	0	0	0	0	0	4	0	0	0	4
20 – 0	0	0	0	0	0	0	0	0	0	0
Total	3798	198	0	3577	23	4914	237	0	4516	161

values, but they result in abnormally high Mahalanobis scores after the distance-based scoring. Since ProGIP directly checks the Mahalanobis scores while detection-only Ranger checks the activation values, such faults can only be detected by ProGIP.

5.3 Analysis of Not a Number (NaN) and Infinity Values

Equation (3) shows the scoring function of Mahalanobis where $f(x)$ is the logits and $\hat{\Sigma}_c^{-1}$ and $\hat{\mu}_c$ are the covariance and mean of logits of ID samples for a class c. In this equation, high logits $f(x)$ can easily produce NaN values in PyTorch when the values are extremely large. In our PyTorch implementation, NaN values always result in a classification output of 0 (first label). Additionally, comparisons in PyTorch involving at least one NaN value always return false. Since the implementation of Mahalanobis considers an inference as OOD if the confidence score is less than or equal to the threshold, it treats a confidence score with NaN as OOD. Such behavior with NaN values cannot be detected without explicit NaN checking, which we have embedded in our ProGIP implementation. We observed that around 58.3% of classification and ID/OOD detection failures (91.1% for ODIN and 25.19% for Mahalanobis) are detected by comparing gradients and ODIN/Mahalanobis scores with δ_{F1} and δ_{F2} based on Equations (4), (5), and (6) rather than detected by NaN handling as resulted in Tables 5 and 6, demonstrating its effectiveness beyond just handling NaN cases.

5.4 Impact of Bit Position on Fault Coverage

While our primary focus was on high-order bit-flips due to their significant impact on neural network outputs, we also analyzed the effect of bit position on fault coverage. As acknowledged in the reviewer feedback, lower and middle-order bit-flips can also cause errors, particularly marginal classification errors where the decision boundaries are close. Tables 5 and 6 show the bit-wise fault injection results for ODIN and Mahalanobis, respectively. Our analysis revealed that faults in the sign bit (bit 31) and the highest exponent bits (bits 30–27) accounted for around 98.49% of the classification and ID/OOD detection failures. Specifically, faults on the highest exponent bit (30) accounted for 92.18% of the total failures.

ProGIP achieved excellent coverage for these high-order bit-flips, with 99.47% detection rates for bits 30–31 and 92.46% for bits 27–29. However, for middle-order bits (26–23), which primarily affect the exponent field of the floating-point representation, ProGIP can only detect 9.18% of critical faults. Further, for lower-order bits (22–0), which mostly affect the mantissa field, ProGIP cannot detect any faults. This lower coverage for mantissa bits is expected, as these bits typically cause

Table 6. Bit-Wise Fault Injection Results for Mahalanobis with and without ProGIP

Faulty bit	Mahalanobis									
	Classification failure					ID/OOD detection failure				
	Number of failures	Detected by ProGIP based on				Number of failures	Detected by ProGIP based on			
		NaN check	Inf check	δ_{F1} check	Undetected		NaN check	Inf check	δ_{F1}, δ_{F2} check	Undetected
31 (sign bit)	2	0	0	0	2	21	0	0	0	21
30	3858	3359	0	495	4	4229	3917	4	272	36
29	4	0	0	4	0	341	0	0	328	13
28	2	0	0	2	0	343	0	0	332	11
27	5	0	0	5	0	243	0	0	229	14
26	2	0	0	0	2	46	0	0	2	44
25	2	0	0	0	2	35	0	0	1	34
24	0	0	0	0	0	14	0	0	0	14
23	0	0	0	0	0	11	0	0	0	11
22	0	0	0	0	0	6	0	0	0	6
21	0	0	0	0	0	14	0	0	0	14
20 – 0	0	0	0	0	0	0	0	0	0	0
Total	3875	3359	0	506	10	5303	3917	4	1164	218

smaller deviations that might not exceed the threshold values used in ProGIP. It's worth noting that despite the lower detection rate for lower exponent bits and mantissa bits, these bits also caused significantly fewer failures overall. In our experiments, bits 26–23 and 22–0 collectively accounted for less than 1.16% and 0.15% of all failures, respectively, limiting the impact of their lower detection rate on the overall effectiveness of ProGIP.

5.5 Ablation Study: Fault Coverage of the First and Second Detectors of ProGIP

To evaluate the fault coverage of each of the two checkers in ProGIP, we analyzed the number of classification and ID/OOD detection failures detected by each detector. Table 7 shows the number of detected failures when (i) only the first detector is enabled with threshold δ_{F1}, (ii) only the second detector is enabled with threshold δ_{F2}, and (iii) both are enabled (full ProGIP). The first detector with δ_{F1} can protect the first forward and backward passes but cannot detect any faults on the second forward pass since it checks the gradient before the second forward pass. On the other hand, the second detector with δ_{F2} covers the second forward pass, while it cannot prevent failures at the first forward and backward passes. This is because the GIP process based on Equation (1) only utilizes the sign of the gradient values and drops the magnitudes of the gradient, as discussed in Section 3.2.2. Thus, the ablation confirms that both detectors are required to achieve near-complete fault coverage.

5.6 Summary of Results

Our experimental results demonstrate that ProGIP achieves comprehensive protection against soft errors in neural networks with GIP-based OOD detection methods, with several key advantages:

- **High fault coverage:** ProGIP detects 97.70% of critical failures across different neural networks, OOD detection methods, and execution passes, providing robust protection against soft errors.
- **Minimal runtime overhead:** With just 0.84% average runtime overhead, ProGIP is significantly more efficient than the detection-only Ranger, which incurs 135.07% overhead.
- **Comprehensive protection:** ProGIP effectively protects all three execution passes of GIP solutions (first forward, backward, and second forward).
- **Adaptability:** ProGIP works effectively with different GIP-based OOD detection methods like ODIN and Mahalanobis and neural network architectures such as DenseNet-BC and ResNet34, demonstrating its versatility.

Table 7. Fault Injection Results of Individual ProGIP Checkers and Complete ProGIP with Both Checkers

OOD detection method	Faulty pass	Classification failure				ID/OOD detection failure			
		Number of failures	Detected by			Number of failures	Detected by		
			δ_{F1} only check	δ_{F2} only check	ProGIP (δ_{F1}, δ_{F2})		δ_{F1} only check	δ_{F2} only check	ProGIP (δ_{F1}, δ_{F2})
ODIN	First forward	3798	3775 (99.39%)	0 (0.00%)	3775 (99.39%)	650	630 (96.92%)	0 (0.00%)	630 (96.92%)
	Backward pass	-	-	-	-	600	552 (92.00%)	0 (0.00%)	552 (92.00%)
	Second forward	-	-	-	-	3664	0 (0.00%)	3571 (97.46%)	3571 (97.46%)
	Total	3798	3775 (99.39%)	0 (0.00%)	3775 (99.39%)	4914	1182 (24.05%)	3571 (72.67%)	4753 (96.72%)
Maha	First forward	3875	3865 (99.74%)	0 (0.00%)	3865 (99.74%)	1142	1041 (91.16%)	0 (0.00%)	1041 (91.16%)
	Backward	-	-	-	-	1131	1105 (97.70%)	0 (0.00%)	1105 (97.70%)
	Second forward	-	-	-	-	3030	0 (0.00%)	2939 (97.00%)	2939 (97.00%)
	Total	3875	3865 (99.74%)	0 (0.00%)	3865 (99.74%)	5303	2146 (40.46%)	2939 (55.42%)	5085 (95.89%)

These results confirm that ProGIP provides an efficient and effective solution for protecting GIP-based OOD detection approaches from soft errors, significantly enhancing the reliability of deep learning systems in real-world environments.

6 Applicability: Activation Modification Approaches with ProGIP

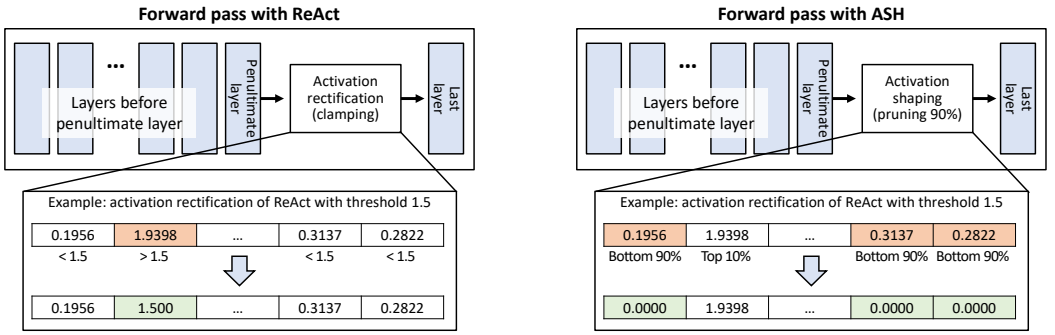
To evaluate the applicability of ProGIP for other OOD detection solutions, this section examines how ProGIP can be extended to two representative activation-modification OOD detectors: ReAct [45] and ASH [8].

6.1 ReAct and ASH

ReAct [45] rectifies the activation values at the penultimate layer by clamping the activation values larger than a certain threshold to the threshold. This activation modification is based on the observation that activation values of OOD inputs at the penultimate layer tend to be biased, i.e., few values have sharp positive values, while the ones of ID inputs are usually well-distributed. Therefore, the activation rectification of ReAct can decrease the confidence of OOD samples while affecting the confidence of ID samples less. Figure 6(a) shows an example of the activation rectification of ReAct with a fixed threshold value of 1.5.

On the other hand, ASH [8] shapes the activation values at a late layer such as penultimate layer by removing a large portion, e.g., 90% of activation values based on a simple top-K criterion and adjusting the modified activation values for sparsification. ASH provides multiple options to adjust the modified activations, and we utilize ASH-P (prunning), which just removes the low activation values and does nothing since it provides better results in our environment compared to other adjustment approaches. Figure 6(b) shows an example of the activation shaping of ASH with pruning the bottom 90% of activation values at the penultimate layer.

ReAct and ASH can utilize various OOD scoring, such as the softmax score [18], energy score [33], and ODIN [32]. In this section, we apply ProGIP to ODIN + ReAct based on their official implementation [26], which performs the ODIN scoring with GIP but rectifies the activation values at the penultimate layers of the first and second forward passes. On the other hand, we apply ProGIP to Softmax score + ASH without GIP to follow their official implementation [7] and showcase the adaptability of ProGIP for non-GIP solutions. For Softmax score + ASH, we observed that the



(a) Example of ReAct [45] activation rectification.

(b) Example of ASH [8] activation shaping.

Fig. 6. Both ReAct and ASH modify the activation values at the penultimate layer of the forward pass.

confidence of softmax scores, i.e., maximum softmax value, usually becomes 1, so we applied the temperature scaling similarly to ODIN. Therefore, Softmax score + ASH works similar to the ODIN but with the activation shaping of ASH and without GIP.

6.2 Applying ProGIP to the Activation Modification Approaches

Applying ProGIP to the OOD detection approaches with the activation modification, such as ReAct and ASH, should carefully add additional detector(s) before the activation modification. This is because clamping or pruning can mask the very outliers that ProGIP relies on; fault symptoms can disappear before the detection. For example, a fault on a high-order bit in ODIN + ReAct can result in abnormally high activation values, but such high values are clamped as the threshold value of ReAct at the penultimate layer so that ODIN cannot detect such a fault. Note that such clamping hides the symptoms of faults, but does not eliminate the effects of faults; the faulty neuron can propagate the effect of the fault for multiple outputs after forwarding one layer, and the neurons affected by the fault propagation are replaced with the threshold values by the clamping, losing their original values with information.

To protect ODIN + ReAct with GIP, ProGIP adds the first fault detector that checks the gradient after the backward pass, the second fault detector that checks the ODIN score after the second forward pass, and the scoring function based on Equations (4) and (5). Further, ProGIP adds extra fault detectors that check abnormally high values in the activation values right before the rectification of ReAct, i.e., clamping, at the penultimate layers of the first and second forward passes. On the other hand, since Softmax score + ASH does not adopt GIP, it has just one forward pass without extra forward or backward passes. Therefore, ProGIP adds one fault detector that checks the softmax score with temperature scaling based on Equation (5) and another fault detector that checks abnormally high values in the activation values right before the activation shaping of ASH, i.e., pruning, at the penultimate layer.

6.3 Experiments for ASH and ReAct

To evaluate the fault coverage of ProGIP for ReAct and ASH, we applied Ranger and ProGIP to ODIN + ReAct and softmax score + ASH and conducted fault injection experiments for them. Specifically, we apply ProGIP without or with extra detectors before the activation modification to evaluate the effectiveness of the extra detectors for activation modification approaches. We utilize DenseNet-BC [21], ResNet-34 [55], and MobileNetV2 [41] models in Section 4.1 and the fault injection setup described in Section 4.3. For ODIN + ReAct with GIP, we selected the clamping

Table 8. Fault Injection Results of ProGIP and Detection-Only Ranger for ReAct and ASH

OOD detection and activation modification	Method	Fault injection runs	Originally correct run*	Classification failure	ID to OOD detection failure	OOD to ID detection failure
ODIN + ReAct with GIP	Unprotected (Original)	1,800,000	1,493,970	3,651	4,885	451
	Detection-only Ranger			19	52	63
	ProGIP without checkers before activation modification			704	3845	323
	ProGIP with checkers before activation modification			23	105	69
Softmax score + ASH without GIP	Unprotected (Original)	600,000	471,560	3,962	28	3,199
	Detection-only Ranger			17	20	64
	ProGIP without checker before activation modification			22	28	80
	ProGIP with checker before activation modification			22	28	80
* The execution that can produce correct classification and ID/OOD results if no fault is injected.						

threshold as 1.5 for DenseNet-BC and ResNet-34 and 0.5 for MobileNetV2 by profiling the ID vs OOD AUROC results between thresholds 0.5, 1.0, and 1.5. For softmax score + ASH, we use ASP-P90, which prunes the 90% bottom of the activation values at the penultimate layer.

We injected 100,000 faults for each datatype (ID/OOD) and pass, including the first forward, backward, and second forward passes of ODIN + ReAct with GIP and the forward pass of Softmax score + ASH without GIP. We executed a total of 1.8M fault injection runs—100,000 injections × 2 (data types) × 3 (passes) × 3 (networks) for ReAct and a total of 600k faults–100,000 injections × 2 (data types) × 1 (pass) × 3 (networks) for ASH.

Table 8 shows the fault injection results of detection-only Ranger and ProGIP with and without checker(s) right before the activation modification for ODIN + ReAct with GIP and Softmax score + ASH without GIP. Several key observations from the fault injection results are as follows:

— Overall, the results show that ProGIP provides fault coverage comparable to detection-only Ranger for the activation modification schemes by adding an additional fault detector right before the activation modification for the forward pass(es).

— ProGIP, without additional checkers right before the activation modification, cannot effectively detect faults for ODIN + ReAct. As discussed in Section 6.2, this is because the activation rectification of ReAct in Figure 6(a) clamps abnormally high values induced by the fault. On the other hand, it provides the same fault coverage as ProGIP with an additional checker right before the activation modification for softmax score + ASH. This is because the activation shaping of ASH keeps the top 10% of the activation values, and therefore, abnormally high values remain after the activation shaping and right before the fault detector of ProGIP for the softmax scores. Considering this observation, ProGIP does not need the additional fault detection right before the activation modification for ASH, while the additional detector is essential for ReAct.

— As shown in the fault injection results for ODIN in Figure 4, high-order bit-flips on the second forward pass tend to result in OOD to ID detection failures in ODIN. However, we observed that high-order bit-flips on the second forward pass tend to result in ID to OOD detection failures in ODIN + ReAct. This is due to the activation rectification of ReAct that clamps higher values. A fault in a high-order bit affects the magnitude of a value. When this value passes a layer, the outputs of the layer that multiplied the faulty value and weights can be either an abnormally high positive value or abnormally low negative value, based on the sign of the fault value

Table 9. Runtime Measurement Results of ProGIP and Detection-Only Ranger for ReAct and ASH

Network	OOD detection	Normalized execution time		
		Detection-only Ranger	ProGIP without additional detector(s) before the activation modification	ProGIP with additional detector(s) before the activation modification
DenseNet-BC [21]	ODIN [32] + ReAct [45]	253.54%	100.83%	101.97%
	Softmax score [18] + ASH [8]	193.70%	100.64%	101.01%
ResNet-34 [55]	ODIN [32] + ReAct [45]	241.12%	100.46%	101.29%
	Softmax score [18] + ASH [8]	179.73%	100.48%	101.24%
MobileNetV2 [41]	ODIN [32] + ReAct [45]	211.37%	101.38%	102.06%
	Softmax score [18] + ASH [8]	165.86%	100.57%	100.88%
Average		207.55%	100.73%	101.41%

and weights. At the penultimate layer, the activation rectification of ReAct clamps abnormally high positive values, but all of the negative values and other values affected by the abnormally negative values remain the same. Consequently, high-order bit-flips in ODIN + ReAct tend to decrease the confidence scores, while they tend to increase the confidence scores of the original ODIN without ReAct.

We also measured the runtime of ODIN + ReAct and softmax score + ASH with the detection-only Ranger and ProGIP with the same setup in Section 4.4. Table 9 shows the runtime measurement results. Similar to the evaluation in Section 5.1, ProGIP only incurs 0.73% average runtime overhead without additional checker(s) for the activation modification and 1.41% average runtime overhead with additional checker(s) for the activation modification. In Table 9, the detection-only Ranger for softmax score + ASH shows lower overhead compared to the detection-only Ranger for ODIN + ReAct. Note that softmax score + ASH does not proceed with GIP; it only has a forward pass, while ODIN + ReAct incurs the GIP process and, therefore, executes two forward passes and one backward pass. This runtime overhead gap implies a higher runtime overhead of PyTorch hook-based protection for the backward pass.

7 Related Work: Soft Error Mitigation for Neural Networks

Soft errors are transient faults that result in temporary bit-flip errors in transistors induced by external sources such as alpha particles, thermal neutrons, or cosmic rays [34]. These external sources can cause fluctuations in signal voltage that lead to the flipping of bit values from 1 to 0 or vice versa. These errors pose a significant challenge as they can lead to application malfunction even when the software and hardware components are flawless, making them a central focus in the design of safety-critical systems.

Traditional redundancy schemes against soft errors, such as **dual modular redundancy** (**DMR**), are difficult to apply to neural networks due to their significant hardware cost or runtime overhead. Therefore, soft error protection for neural networks focuses on efficient redundancy mechanisms. For example, **algorithm-based fault tolerance** (**ABFT**) solutions [1, 36] exploit mathematical properties of matrix multiplication operations to introduce efficient checksums. Another approach [43] 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 [3, 12] insert checkers between layers of the target network to detect abnormally high values affected by faults. These studies observed that faults on high-order bits contribute to the majority of soft-error-induced failures in neural networks [3, 12], and such large deviations can be easily detected since they induce significantly large values in layer outputs that rarely appear in the absence of faults [3, 12, 28]. 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.

While high-order bit-flips account for most critical failures and are relatively easy to detect, it's important to acknowledge that lower and middle-order bit-flips can also cause errors. These more subtle faults might lead to marginal classification errors where the output still appears reasonable but is incorrect. Traditional range-based detection methods may not effectively identify these types of errors, as the resulting values often remain within expected ranges. This limitation is particularly relevant in classification tasks with fine decision boundaries, where even small deviations can cause misclassifications.

Detecting soft errors in OOD detection solutions is crucial since soft errors in neural networks not only affect the classification result but can also change the ID/OOD detection result. Only a few solutions [11, 42] have the potential to provide holistic reliability against non-malicious threats, including both hardware faults and OOD inputs, by considering soft errors (or permanent faults) and OOD as anomalies. However, such solutions cannot distinguish between faults and OOD inputs, which is crucial for system designers who want to establish appropriate mitigation solutions for each anomaly type.

While previous works have made significant progress in addressing either OOD detection or soft error mitigation, there remains a gap in effectively combining both approaches, especially for neural networks using GIP methods. Our work addresses this gap by proposing ProGIP, which specifically targets the protection of GIP approaches from soft errors with minimal overhead.

8 Conclusion and Future Directions

In this article, we presented ProGIP, a novel approach to protect GIP methods for OOD detection from soft errors. As deep neural networks become increasingly integrated into safety-critical systems, ensuring their reliability against non-malicious threats such as OOD inputs and soft errors is paramount. ProGIP addresses this challenge by distinguishing between ID inputs, OOD inputs, and fault-affected inferences with minimal overhead. Our key insight was to analyze how high-order bit-flips affect different execution passes of GIP solutions and to strategically place fault detectors at critical points in the computation pipeline. By inserting just two software-level range-based checkers, ProGIP achieves 97.7% fault coverage while incurring only 0.84% runtime overhead. We conducted extensive experiments with 3.6 million fault injections across three neural network architectures and two OOD detection methods. The results demonstrated that ProGIP effectively protects against both classification failures and ID/OOD detection failures across all execution passes, significantly enhancing the reliability of GIP-based OOD detection. Compared to a detection-only implementation of Ranger, ProGIP achieves comparable fault coverage with significantly lower overhead (0.84% vs. 135.07%). This efficiency makes ProGIP particularly suitable for real-time applications where execution speed is critical. ProGIP represents an important step toward developing holistic reliability solutions for deep learning systems that must operate in challenging real-world environments. By enabling the distinction between different types of threats, ProGIP allows system designers to implement appropriate countermeasures for each scenario, such as collecting OOD inputs for future training or adjusting system parameters to reduce soft error rates.

ProGIP currently addresses soft errors only in the classification context. Future work might expand the scope of our technique to segmentation or object detection contexts. For example, per-pixel logits and gradient maps in segmentation networks behave like the class logits in classification. Therefore, range checks on these maps (e.g., detecting NaN, infinity values, or huge gradient magnitudes) would catch faults that corrupt region predictions. On the other hand, object detection pipelines produce bounding-box scores and class confidences. Two-point detectors of ProGIP

could first validate gradients used in box refinement (backward pass) and then threshold the final confidence scores for each proposed box.

The coarse-grained fault detection of ProGIP enables lightweight fault detection. However, it also makes instant fault detection, such as Ranger [3], hard for ProGIP. This is because a faulty neuron with an abnormally high value propagates the faulty value to all output neurons for a layer where the faulty neuron is an input neuron, except for output neurons that have zero weight from the faulty neuron. If the instant fault correction approach (e.g., assigning a specific value such as zero [37] or bounding the range of value [3]) detects the fault after this propagation, the approach will incorrectly correct the fault-propagated neurons, which will lose the original information of multiple neurons in the layer. A coarse-grained fault detection scheme such as ProGIP can only recover such cases by re-executing the faulty inference. In other words, ProGIP trades off the fault-detection latency to minimize the runtime overhead to protect the target network, while Ranger utilizes the instant fault detection latency for the correction. Future work might suggest a better fault recovery scheme that fits with ProGIP.

Finally, we used an empirical 200% margin for ProGIP, while the margin can be narrowed for each network, OOD detection scheme, and fault detector. For example, we observed that ODIN scores in our ID and OOD test never exceeded the maximum ODIN score profiled from the ID train set. Still, we do not have a good methodology to select a minimum margin that can prevent false detection without an extra test set. A methodology to select the proper margin without any test set can improve the fault coverage of ProGIP.

References

[1] Sandeep Bal, Chandra Sekhar Mummidi, Victor Da Cruz Ferreira, Sudarshan Srinivasan, and Sandip Kundu. 2023. A novel fault-tolerant architecture for tiled matrix multiplication. In *Proceedings of the 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[2] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308. DOI: https://doi.org/10.1109/JETCAS.2019.2910232

[3] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. 2021. A low-cost fault corrector for deep neural networks through range restriction. In *Proceedings of the 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–13.

[4] Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2019. 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*. 1–23.

[5] Charles Corbière. 2022. Robust deep learning for autonomous driving. Ph.D. Dissertation. HESAM Université, Paris, France. NNT: 2022HESAC032. tel-04351564f.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.

[7] Andrija Djurisic. 2022. Retrieved September 9, 2025 from https://github.com/andrijazz/ash

[8] Andrija Djurisic, Nebojsa Bozanic, Arjun Ashok, and Rosanne Liu. 2023. Extremely simple activation shaping for out-of-distribution detection. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://arxiv.org/abs/2209.09858

[9] Yicong Dong, Rundong He, Guangyao Chen, Wentao Zhang, Zhongyi Han, Jieming Shi, and Yilong Yin. 2025. G-OSR: A comprehensive benchmark for graph open-set recognition. arXiv:2503.00476. Retrieved from https://arxiv.org/abs/2503.00476

[10] Farshad Firouzi, Mostafa E. Salehi, Fan Wang, and Sied Mehdi Fakhraie. 2011. An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects. *Microelectronics Reliability* 51, 2 (2011), 460–467.

[11] Gabriele Gavarini, Diego Stucchi, Annachiara Ruospo, Giacomo Boracchi, and Ernesto Sanchez. 2022. Open-set recognition: An inexpensive strategy to increase DNN reliability. In *Proceedings of the 2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 1–7.

[12] Behnam Ghavami, Mani Sadati, Zhenman Fang, and Lesley Shannon. 2022. FitAct: Error resilient deep neural networks via fine-grained post-trainable activation functions. In *Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1239–1244.

[13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
[14] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1321–1330.
[15] Shafinul Haque, Andy Wei Liu, Serena Liu, and Jonathan H. Chan. 2021. 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*. 1–7.
[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[17] Yi He, Prasanna Balaprakash, and Yanjing Li. 2020. Fidelity: Efficient resilience analysis framework for deep learning accelerators. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 270–281.
[18] Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
[19] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2014. Distilling the knowledge in a neural network. In *Proceedings of NeurIPS 2014 Workshop on Deep Learning and Representation Learning*. Retrieved from http://arxiv.org/abs/1503.02531
[20] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. 2020. 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*. 10951–10960.
[21] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.
[22] Wenyu Jiang, Yuxin Ge, Hao Cheng, Mingcai Chen, Shuai Feng, and Chongjun Wang. 2023. Read: Aggregating reconstruction error into out-of-distribution detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 14910–14918.
[23] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. Master's thesis. University of Toronto, Toronto, Ontario.
[24] kuangliu. 2018. Retrieved September 9, 2025 from https://github.com/kuangliu/pytorch-cifar/blob/master/models/mobilenetv2.py
[25] Zeqiang Lai, Kaixuan Wei, Ying Fu, Philipp Härtel, and Felix Heide. 2023. ∇-prox: Differentiable proximal algorithm modeling for large-scale optimization. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–19.
[26] CS Research Group led by Prof. Sharon Li. 2021. Retrieved September 9, 2025 from https://github.com/deeplearning-wisc/react
[27] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in Neural Information Processing Systems* 31 (2018), 7167–7177.
[28] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. 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*. 1–12.
[29] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2018. Tensorfi: A configurable fault injector for tensorflow applications. In *Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE.
[30] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. arXiv:2006.15704. Retrieved from https://arxiv.org/abs/2006.15704
[31] Yixuan Li. 2018. Retrieved Retrieved September 9, 2025 from https://github.com/facebookresearch/odin
[32] Shiyu Liang, Yixuan Li, and R. Srikant. 2018. Enhancing the reliability of out-of-distribution image detection in neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
[33] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. 2020. Energy-based out-of-distribution detection. *Advances in Neural Information Processing Systems* 33 (2020), 21464–21475.
[34] Shubhendu S. Mukherjee, Joel Emer, and Steven K. Reinhardt. 2005. The soft error problem: An architectural perspective. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. IEEE, 243–247.
[35] Hyungjun Oh, Junyeol Lee, Hyeongju Kim, and Jiwon Seo. 2022. Out-of-order backprop: An effective scheduling technique for deep learning. In *Proceedings of the 17th European Conference on Computer Systems*. 435–452.
[36] Elbruz Ozen and Alex Orailoglu. 2019. Sanity-check: Boosting the reliability of safety-critical deep neural network applications. In *Proceedings of the 2019 IEEE 28th Asian Test Symposium (ATS)*. IEEE, 7–75.
[37] Elbruz Ozen and Alex Orailoglu. 2020. Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.
[38] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. 2021. Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–38.

[39] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I. August. 2005. SWIFT: Software implemented fault tolerance. In *Proceedings of the International Symposium on Code Generation and Optimization*. IEEE, 243–254.

[40] Lukas Ruff, Jacob R. Kauffmann, Robert A. Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. 2021. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE* 109, 5 (2021), 756–795.

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[42] Christoph Schorn and Lydia Gauerhof. 2020. Facer: A universal framework for detecting anomalous operation of deep neural networks. In *Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–6.

[43] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Efficient on-line error detection and mitigation for deep neural network accelerators. In *Proceedings of the 37th International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2018*. Springer, 205–219.

[44] Muhammad Shafique, Alberto Marchisio, Rachmad Vidya Wicaksana Putra, and Muhammad Abdullah Hanif. 2021. Towards energy-efficient and secure edge AI: A cross-layer framework ICCAD special session paper. In *Proceedings of the 2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[45] Yiyou Sun, Chuan Guo, and Yixuan Li. 2021. React: Out-of-distribution detection with rectified activations. *Advances in Neural Information Processing Systems* 34 (2021), 144–157.

[46] Yongqiang Tian, Shiqing Ma, Ming Wen, Yepang Liu, Shing-Chi Cheung, and Xiangyu Zhang. 2021. To what extent do DNN-based image classification models make unreliable inferences? *Empirical Software Engineering* 26, 5 (2021), 84.

[47] Sachin Vernekar, Ashish Gaurav, Vahdat Abdelzad, Taylor Denouden, Rick Salay, and Krzysztof Czarnecki. 2019. Out-of-distribution detection in classifiers via generation. In *NeurIPS 2019 Workshop on Safety and Robustness in Decision Making*. Retrieved from https://arxiv.org/abs/1910.04241

[48] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L. Willke. 2018. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 550–564.

[49] Zishen Wan, Aqeel Anwar, Abdulrahman Mahmoud, Tianyu Jia, Yu-Shun Hsiao, Vijay Janapa Reddi, and Arijit Raychowdhury. 2022. Frl-fi: Transient fault analysis for federated reinforcement learning-based navigation systems. In *Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 430–435.

[50] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620.

[51] Donghun Yang, Kien Mai Ngoc, Iksoo Shin, Kyong-Ha Lee, and Myunggwon Hwang. 2021. Ensemble-based out-of-distribution detection. *Electronics* 10, 5 (2021), 567.

[52] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. 2024. Generalized out-of-distribution detection: A survey. *International Journal of Computer Vision* 132, 12 (2024), 5635–5662.

[53] Jinsong Zhang, Qiang Fu, Xu Chen, Lun Du, Zelin Li, Gang Wang, XiaoGuang Liu, Shi Han, and Dongmei Zhang 2022. Out-of-distribution detection based on in-distribution data patterns memorization with modern Hopfield energy. In *Proceedings of the 11th International Conference on Learning Representations*.

[54] Jingyang Zhang, Jingkang Yang, Pengyun Wang, Haoqi Wang, Yueqian Lin, Haoran Zhang, Yiyou Sun, Xuefeng Du, Yixuan Li, Ziwei Liu, Yiran Chen, and Hai Li. 2024. OpenOOD v1.5: Enhanced benchmark for out-of-distribution detection. *Journal of Data-centric Machine Learning Research* (2024). https://openreview.net/forum?id=cnnTnJQigsDatasetCertification

[55] Ev Zisselman and Aviv Tamar. 2020. Deep residual flow for out of distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13994–14003.

Received 11 August 2025; revised 11 August 2025; accepted 12 August 2025