Fault injection analysis of Real NVP normalising flow model for satellite anomaly detection

Fault injection analysis of Real NVP normalising flow model for satellite anomaly detection

Gabriele Greco
DET - Politecnico di Torino
Torino, ITA
gabriele.greco@studenti.polito.it

Carlo Cena †
DET - Politecnico di Torino
Torino, ITA
carlo.cena@polito.it

Umberto Albertin
DET - Politecnico di Torino
Torino, ITA
umberto.albertin@polito.it

Mauro Martini
DET - Politecnico di Torino
Torino, ITA
mauro.martini@polito.it

Marcello Chiaberge
DET - Politecnico di Torino
Torino, ITA
marcello.chiaberge@polito.it

***Abstract*—Satellites are used for a multitude of applications, including communications, Earth observation, and space science. Neural networks and deep learning-based approaches now represent the state-of-the-art to enhance the performance and efficiency of these tasks. Given that satellites are susceptible to various faults, one critical application of Artificial Intelligence (AI) is fault detection. However, despite the advantages of neural networks, these systems are vulnerable to radiation errors, which can significantly impact their reliability. Ensuring the dependability of these solutions requires extensive testing and validation, particularly using fault injection methods. This study analyses a physics-informed (PI) real-valued non-volume preserving (Real NVP) normalizing flow model for fault detection in space systems, with a focus on resilience to Single-Event Upsets (SEUs). We present a customized fault injection framework in TensorFlow to assess neural network resilience. Fault injections are applied through two primary methods: Layer State injection, targeting internal network components such as weights and biases, and Layer Output injection, which modifies layer outputs across various activations. Fault types include zeros, random values, and bit-flip operations, applied at varying levels and across different network layers. Our findings reveal several critical insights, such as the significance of bit-flip errors in critical bits, that can lead to substantial performance degradation or even system failure. With this work, we aim to exhaustively study the resilience of Real NVP models against errors due to radiation, providing a means to guide the implementation of fault tolerance measures.**

***Index Terms*—Fault tolerance, Signal processing, Fault injection, Normalising flow, Space vehicle telemetry.**

I. Introduction

Satellites are essential for fields like communications, navigation, and Earth observation, as demonstrated by numerous missions, such as [1, 2, 3]. As the number of space-based systems grows, so does the need to protect them from faults that can occur during their lifespan and from the harsh radiation environment, requiring robust hardware and software models. Neural Networks (NNs), known for handling large datasets and generating actionable insights, are increasingly being deployed in space missions for this purpose. For instance, ESA's BepiColombo [1] leverages NNs in fault detection to maintain spacecraft health on its journey to Mercury. Among these methodologies, the Physics-Informed Real NVP model introduced by [4] demonstrated its ability to model complex distributions while incorporating domain-specific knowledge in the context of fault detection for an aerospace dataset. However, the complexity of NNs poses challenges for reliability assessment, especially under radiation conditions, as seen in Deep Space 1 (DS-1) [5]. Radiation effects are categorized as cumulative or single-event errors, we will focus on the latter. These are called Single-Event Effects (SEEs), and can be permanent or transient, such as soft errors like Single-Event Upsets (SEUs) and Single-Event Transients, both impacting system reliability. In Deep Neural Networks (DNNs), faults from radiation can alter outputs unpredictably, leading to Silent Data Corruption (SDC) or Detected Unrecoverable Errors (DUE). Although SDCs not affecting detection accuracy may be tolerable, those causing misdetection are critical. Various methodologies exist for evaluating the reliability of computing devices, spanning different abstraction levels. Field Test involves exposing devices to

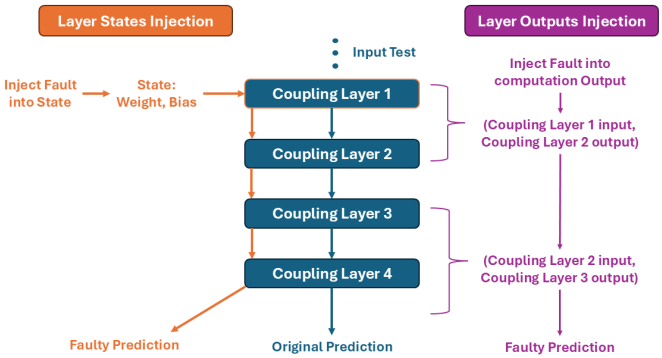


Fig. 1: Representation of Layer States and Outputs injections inside a Real NVP Network.

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 117/2023 and MUR – DM 351/2022.

† Corresponding author

natural particle flux and counting observed errors [6]. Beam Experiment induces faults by interacting accelerated particles with the silicon lattice at the transistor level, providing highly realistic error rates [7]. Microarchitecture-level fault injection offers broader fault coverage compared to software-level injection, as faults can potentially be injected across most system modules [8]. Software fault injection, performed at the highest level of abstraction, has proven effective in identifying code sections that are more susceptible to computational impact when corrupted [9]. Circuit- or gate-level simulations operate at the lowest level of abstraction, inducing either analog current spikes or digital faults while tracking fault propagation [10]. Finally, to enhance efficiency and maintain accuracy, hybrid approaches combining different abstraction levels are often employed [11].

This study prioritizes understanding the impact of soft errors on NNs, through software fault injection, emphasizing the influence of model architecture and resilience rather than focusing on hardware-specific cumulative degradation. The fault injection framework requires careful engineering to avoid unrealistic results, because when evaluating the reliability of complex computing devices executing DNNs, it's crucial to consider that radiation-induced faults originate at the physical transistor level and then propagate through the architecture, ultimately affecting the software and modifying the output. Evaluations closer to the physical layer offer a more realistic perspective, while those closer to the software layer are more efficient [12]. In fact, while offering advantages such as lower costs, better controllability, and easier deployment for developers, the commonly adopted fault model (typically bit flip) [13, 14, 15, 16] may be accurate for main memory structures but less realistic for faults within computing cores or control logic, where the programmer has limited influence. Our approach involves emulating the effects of radiation by modifying memory values and altering key network elements, temporarily disabling them to assess their impact on performance and overall network behavior. To sum-up, we propose a framework for testing Normalizing Flow networks, in particular Real NVP models, under simulated radiation-induced faults, to evaluate model robustness (see Figure 1). It enables simultaneous testing of multiple models with varied hyperparameters under different fault conditions, offering insights into model resilience.

II. Related Works

The resilience and robustness of DNNs have been widely studied by researchers [15, 17]. Before exploring various fault injection frameworks, it is crucial to understand the consequences of soft errors and the impact of perturbations introduced at different components of the network. [15] described the four main parameters that influence the impact of soft errors in DNNs:

1) Topology and Data Type: Each DNN has a unique architecture and different combination of data types, both factors affect error propagation.
2) Bit Position and Value: The sensitivity of each bit position varies, depending on the data type. High-order exponent bits are more likely to cause SDCs when corrupted, while mantissa and sign bits are less critical. Bit-flips from 0 to 1 in higher-order bits are more likely to cause errors than those from 1 to 0, as correct values in DNNs tend to cluster around zero [18].
3) Layers: Errors propagate in distinct ways across different layers depending on their type and position.
4) Data Reuse: Data reuse strategies in DNN accelerators' dataflows impact the Silent Data Corruption (SDC) probability.

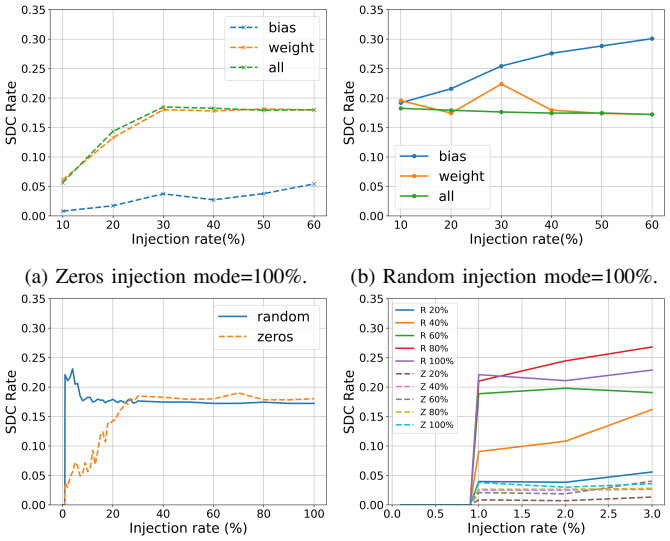
The authors of [15] reach these results by testing several networks, such as AlexNet [19], CaffeNet [20], and NiN [21], using a DNN simulator on various datasets to inject faults and evaluate performance by calculating the SDC Probability and the Failure-in-Time (FIT) rate. When considering layer position and type, [15] has highlighted the non-uniform impact of faults on layers positioned at different depths. Faults occurring in earlier layers are more likely to propagate, but many are masked by operations like pooling or ReLU. Additionally, the numerical magnitude of faulty activations, rather than their quantity, significantly influences the probability of SDC errors. Finally, the above-mentioned paper suggests strategies to improve resilience: (I) DNNs should use a type that offers sufficient numerical range and precision to operate with success, as restricting data types and suppressing dynamic value ranges can help mitigate the effects of bit-flips; (II) normalization layers can improve accuracy by mitigating SDCs, as faulty values are normalized alongside fault-free values. While these insights provide a foundation, they primarily focus on DNNs and may not fully capture the nuances of time-series data. To address this gap, our contribution delves into the resilience of 32-bit floating-point precision Real NVP models designed for high-dimensional time-series data.

III. Methodology

In this section we provide an overview of the neural network architecture and loss, and of the metric used, and describe the considered dataset.

A. *Physics-Informed Real NVP*

Real NVP [22] is a neural network architecture belonging to the class of normalizing flows models, a class of generative models, that transforms a simple probability distribution into a more complex one that matches the data distribution through a sequence of invertible functions. Real NVP introduces non-volume-preserving transformations, which offer more flexibility in modeling complex distributions while still allowing efficient computation of the determinant of the Jacobian, which is necessary for calculating densities. Coupling Layers are the fundamental component of these networks, built as a flexible and tractable bijective function composed of two Fully-Connected (FC) neural networks that use half of the input, which remains unaltered, to compute respectively a scale and a translation factor to be applied to the remaining half



(a) Zeros injection mode=100%.

(b) Random injection mode=100%.

(c) Injections from 0 to 100 layers percentage.

(d) Start of performance degradation for Zeros (Z) and Random (R).

Fig. 2: Zeros and Random Layer States injections.

of the input. [4] enhanced this architecture with a physics-informed loss created by extracting relationships from the considered dataset, ADAPT [23].

B. *Dataset*

We used the same dataset and configuration used by [4]. The training and testing phases of the NN models are performed with the ADAPT dataset [23], an Electrical Power System (EPS) dataset. In particular, given the dataset, we created 3 splits. This allows us to train three models with the same hyper-parameters on three distinct training sets, which are then evaluated on their corresponding test sets. The final score for a specific metric is calculated as the average of the scores obtained from each model. Therefore, when we refer to the results of a model, we mean the average across these different splits.

C. *Fault Injections*

This section explores the layer states and layer outputs injection processes (Figure 1) to assess the impact of targeted modifications on neural network behavior. These processes are controlled by several parameters, allowing for a wide range of configurations. The parameters used to evaluate the model behavior when injecting the layer states are:

- **Mode**: Specifies the percentage of layers to be injected, with options ranging from 20% to 100%.
- **Variable**: Determines the type of variable targeted between bias, weight, or both of them at the same time, defined as 'all' in the plots.
- **Type**: Defines the injection method by Zeros (sets variables to zero), Random (applies Gaussian noise), and Bit-flips (flips specific bits in the values).
- **Amount**: Sets the injection rate for each layer, expressed as a percentage.
- **Bit**: Sets a specific bit position to be flipped or a random one.
- **Direction**: Used only for bit-flips, it controls the flipping direction (0 to 1, 1 to 0, or both).
- **Sign**: Limits bit-flips applied to the sign bit to positive, negative, or both types of values.

In the Layer Outputs experiments, we used several of the above-mentioned parameters (Type, Amount, Bit, Direction, and Sign) along with:

- **Mode**: Specifies the coupling layer to inject, either as a specific layer, a random layer, or all layers.
- **Variable**: Targets either "scale", "translation", or both types at the same time.
- **Activation**: Chooses the activation functions to be used as targets for injection among ReLU, used in all hidden layers, Tanh or Linear, used respectively in the last layers of the scale and translation networks, or all of them, referenced as "all" in the plots.
- **Method**: Given one or more coupling layers, it determines the injection location by picking "Partial" (final FC layers) or "Complete" (all FC layers).

The bit-flip operation pertains to the manipulation of individual bits in the IEEE 754 [24] single-precision floating-point format (binary32), a widely used 32-bit representation for numerical values. This format consists of a 1-bit sign (S), an 8-bit exponent (E), and a 23-bit significand (F) (with an implicit leading 1), enabling a dynamic range of values defined by (3).

$$VALUE = (-1)^S \cdot 2^{(E-127)} \cdot (1.F) \tag{1}$$

This structure ensures approximately 6 to 9 significant decimal digits of precision. Special bit patterns in the exponent define unique cases, such as de-normalized numbers or zero (E = 0) and infinity/NaN (E = 255), which are critical in numerical computations. Fault injections in layer states enable us to evaluate the model's ability to handle corrupted weights and biases. Injection in layer outputs reveals differences in the resilience of scale and translation operations.

D. *Metric*

To assess model robustness under fault injection, we used a resilience metric, the SDC rate, which evaluates the fraction of injected faults leading to silent data corruptions. The SDC rate is defined in (1).

$$SDC_{rate} = \frac{\sum^{N_{exps}} \sum^{N_{seeds}} SDC_{rate\_exp}}{N_{exps} \cdot N_{seeds}} \tag{2}$$

Where N_{seeds} is the number of different seeds, N_{exps} is the number of experiments executed for each seed, and SDC_{rate_exp} is the SDC rate of each experiment, defined in (2).

$$SDC_{rate\_exp} = \frac{1}{N_{samples}} \sum_{i=1}^{N_{samples}} \begin{cases} 1, \text{ if } \mathbf{x}_i \text{ misclassified} \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

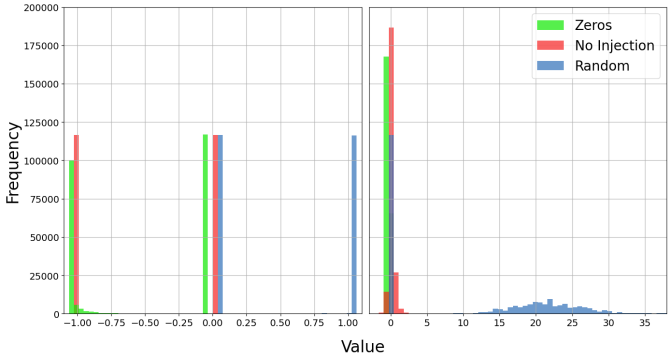


Fig. 3: Distribution of scale (Right) and translation (Left) masked output of the last layer of the network on three different injection configurations.

Where **x** is the vector of $N_{samples}$ samples correctly classified by the Real NVP when no faults were injected. By selectively injecting faults at specific network layers or activation functions, our approach offers both model-level and variable-level insights into the resilience of machine learning models, aligning with metrics like PVF [25] to enhance our understanding of model reliability. The SDC rate is calculated using two approaches: the absolute SDC metric, which evaluates all models against a fixed set of correctly classified test samples, and the relative SDC rate, computed individually for each model based on its unique accuracy and test subset. The absolute SDC provides consistent comparisons across models, while the relative SDC accounts for model-specific performance variations.

IV. Experimental Results

A. Setup

Given a pre-trained model, we performed multiple experiments based on the parameter N_{exps}, which we set equal to 10 to ensure a reliable SDC rate, with $N_{seeds} = 3$ random seeds. For each experiment, the NN state was reset to its original state after every iteration, and the scores for each metric were averaged across all experiments and seeds. For the single-model analysis, presented in sections IV-B and IV-C, we used the smallest model, configured with 4 coupling layers with 3 FC layers each, and 32 units per FC layer. Bit-flip injections in the layer states were performed with a fixed injection rate of 10%, targeting all layers ("Mode" = 100%). For the multiple-model analysis, presented in section IV-D, we evaluated 18 different models trained using various hyper-parameter combinations. These combinations included the number of coupling layers (4 or 6), of FC layers for each coupling layer (3, 4, or 5), and the number of units in each FC layer (32, 48, or 64).

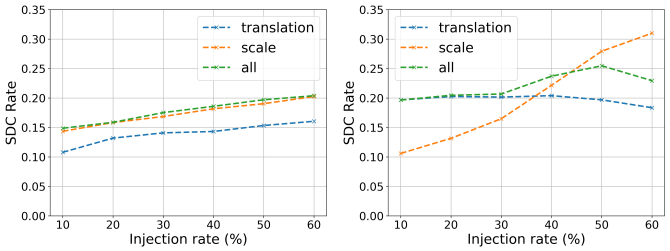
B. Zeros & Random

Layer States: Figure 2 reveals distinct effects of fault injections on the Bias and Weight variables of the Real NVP network, collectively referred to as "all". When injecting faults into all layers, the SDC rate differs significantly depending on the injection type and the variables targeted. Injecting zero values into the bias results in minor performance degradation. Removing it does not disrupt the relative weighting of the layers' outputs, as it primarily adjusts their shift. In contrast, injecting random noise into the bias causes greater degradation, introducing global perturbations to the output and disrupting the controlled behavior of the activation functions. Injecting zeros into the weights causes significant degradation, as it effectively cuts off specific connections within the network, leading to a collapse in learned representations by pruning parts of the network. On the other hand, injecting random noise into the weights has a less drastic impact. The redundancy and distributed nature of weight connections mean that noise often affects only a subset of weights, allowing the model to mitigate the disruption through averaging effects. Performance degradation begins at an injection rate of approximately 0.8% (Figure 2d), and as injection rates increase we observed a saturation in performance (Figure 2c): Zeros injections plateau around a 30% injection rate, while Random injections saturate earlier, at approximately 10%. This behavior stems from the network's internal architecture and the use of tanh activations in scale layers and linear activations in translation layers. Due to these factors, the model can rely on the input alone to drive the output, as the input influences the propagation and transformation of values through the network, even when all variables are zeroed or randomized. Detailed analysis in Figure 3 shows that:

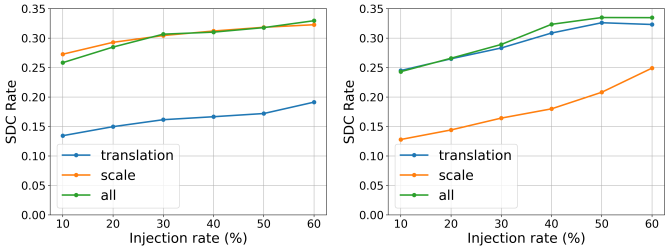
- **Zero injections** result in scale layer outputs oscillating between -1 and 0, with values saturating near -1 due to the tanh activation. Translation layers shift outputs to higher values, with ReLU modifying only negative inputs.
- **Random injections** confine masked scale outputs near 0 or 1, while translation layers produce higher deviations from expected outputs.

These behaviors, linked to activation functions and the propagation of masked operations, reveal critical insights into the network's robustness.

Layer Outputs: This analysis examines the impact of fault injections on layer outputs, focusing on the internal network behavior with unaltered variables. In Figure 4 for "Partial" injections, both random and zero-value faults resulted in linear performance degradation, with translation layers showing a greater resilience. For "Complete" injections, scale layers were robust against random noise but were the most vulnerable to zero-value faults at higher injection rates. Random injections strongly impacted scale layers in "Partial" configurations and translation layers in "Complete" configurations. Activation functions also played a critical role (Figure 5). Random injections severely affected translation layers, especially the final linear FC layers, while scale layers exhibited greater resilience. Zero-value injections caused exponential degradation in ReLU-activated scale layers, while translation layers remained steady across injection rates.

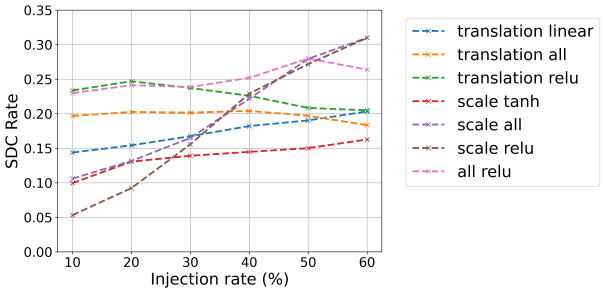


(a) Zeros injection "Partial". (b) Zeros injection "Complete".

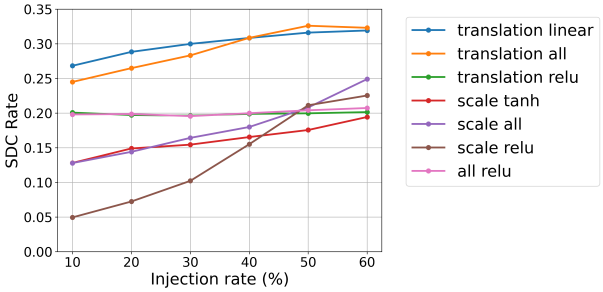


(c) Random injection "Partial". (d) Random injection "Complete".

Fig. 4: Zeros and Random Layer Outputs injections in all layers.



(a) Zeros injections at different activations in all layers.



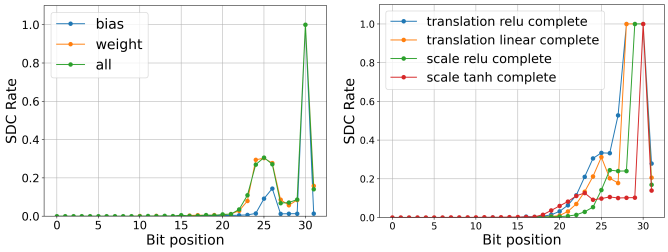
(b) Random injections at different activations in all layers.

Fig. 5: Layer Outputs injections on activation functions.

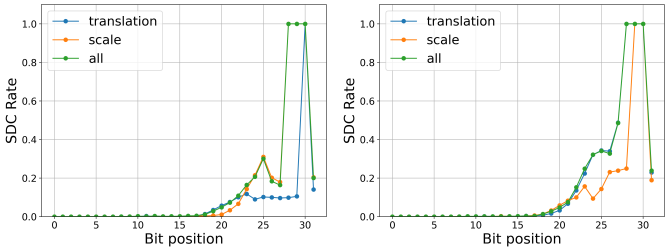
C. *Bit-flips*

Layer States: In Figure 6a, we observe the injection of bit-flips in all directions and in both positive and negative variables. Performance degradation begins at the 20th bit and follows a Gaussian distribution, peaking at the 25th bit in the exponent region, reflecting the critical role of specific bit positions in the Real NVP numerical stability. This finding aligns with the distribution of variables shown in Figure 7, emphasizing the importance of understanding bit-level behavior in designing fault-tolerant systems, especially for safety-critical applications. Flips in the mantissa bits show no performance degradation, while flips in bias have minimal impact, indicating the dominant influence of weights. However, the exponent bits (24–30) present key vulnerabilities due to their distribution. For instance, bit 30 is always set to 0, and flipping it to 1 can cause catastrophic transitions, turning values into infinity or NaN, leading to severe computational errors. This behavior is critical, as experimental data reveal that most fatal faults arise from such flips. Finally, an analysis of the weights showed that they have a slightly higher proportion of negative values, making the network marginally more sensitive to flips affecting these values. Flips from 1 to 0 generally cause minimal degradation and no system failures, whereas flips from 0 to 1 result in slightly higher performance degradation. These findings underscore the necessity of considering bit-level precision to mitigate the risks posed by such errors.

Layer Output: Figures 6d and 6c examined the effects of bit-flips on scale and translation layers, both individually and in combination ('all'). For "Partial" injections, the model's performance was closely aligned with the behavior of scale layers, while "Complete" injections exhibited behavior associated with translation layers. Consistent with prior findings from Zeros and Random injections, translation layers showed resilience when injections occurred in the final layer after linear activation, whereas scale layers were more robust when "Complete" injections targeted hidden layers. Figure 6b shows the failure rates across various activation functions, finding that the scale tanh activation function was the most robust. These results emphasize the impact of model size, depth, and activation functions on bitflip-induced failures, providing insights into the Real NVP vulnerabilities and the effectiveness



(a) Layer States injection with all directions and signs. (b) Layer Outputs injection in all layers at different Activations.



(c) Layer Outputs injection "Partial" in all layers. (d) Layer Outputs injection "Complete" in all layers.

Fig. 6: Bit-flips injections.

of different configurations in mitigating such errors.

D. Multiple Models

This analysis extends the investigation of injection effects to the 18 models referenced in section IV-A. Radial charts (Figures 8 and 9) were employed to compare model performance across different injection scenarios. In these plots we use ”D” and ”U” to represent respectively the number of FC layers in each coupling layer and the number of units of each FC layer. Circular plots indicate consistent performance among the models, whereas deviations highlight variability in response. Figures 10, 11, and 12 present the complete results of the fault injection experiments conducted on all models in parallel coordinates plots. In these figures, darker lines represent higher SDC rates, indicating worse performance. The names of the axes refer to the parameters introduced in III-C.

Layer States: Figures 8 and 10 show that Random injections lead to higher SDC rates with respect to Zeros. The changes in layer states using the Random approach show similar behaviors across all the experiments, leading to a higher SDC rate. In contrast, the Zeros approach yields a lower SDC rate compared to random. Notable results are observed when examining the ”Variable” axis, which corresponds to changes in the weights, biases, or both. Similar behaviors are shown using the Random approach, whereas small variations are observed for the Zeros approach. Weight changes produce a higher SDC rate compared to bias changes, and the highest SDC rate is observed when both variables are altered simultaneously. For zero-value injections, models with deeper layers and fewer coupling units exhibited poorer performance, especially as the layer injection percentage increased. However, increasing the number of coupling units mitigated this degradation (Figures 8a and 8b). Bias injections had minimal impact on performance except in deep models with fewer units, while weights were more susceptible to performance loss, particularly in models with a higher number of units. When both bias and weights were injected, small changes in performance were observed for deep models with a lower amount of units. This highlights the importance of balancing model depth and the number of units to ensure robust performance under faults. For random-value injections similar trends were observed. Models with 4 coupling layers displayed chaotic performance at certain injection rates, such as 40%, indicating sensitivity to small perturbations. Conversely, models with 6 coupling layers showed stable performance except at 40%, where deeper configurations performed worse. Bias injections caused negligible effects across models, except for those with a large number of units.

Layer Outputs: Similar behaviors are observed when com-

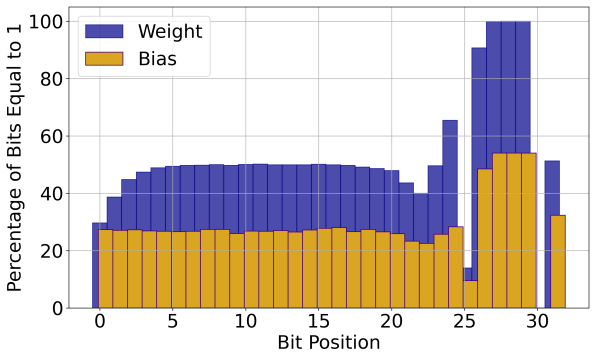
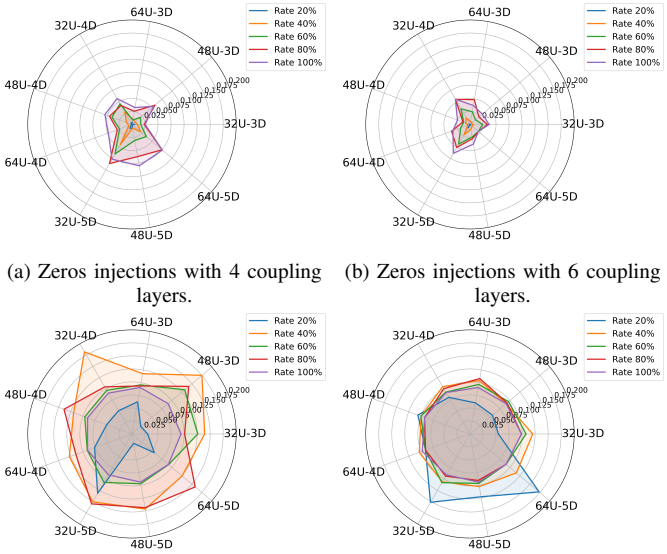


Fig. 7: Distribution of bits equal to 1 for each bit position in Physics-Informed Real NVP Model variables.



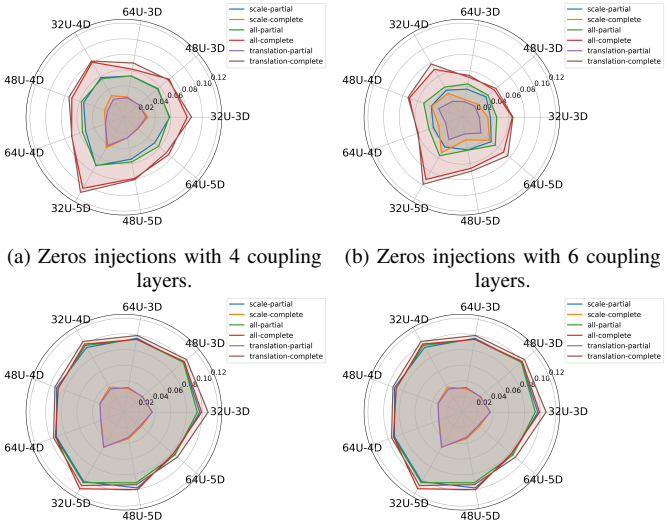
(a) Zeros injections with 4 coupling layers.

(b) Zeros injections with 6 coupling layers.

(c) Random injections with 4 coupling layers.

(d) Random injections with 6 coupling layers.

Fig. 8: Layer States injection across 18 Models with a fixed injection rate of 10% and different variable layer percentage.



(a) Zeros injections with 4 coupling layers.

(b) Zeros injections with 6 coupling layers.

(c) Random injections with 4 coupling layers.

(d) Random injections with 6 coupling layers.s

Fig. 9: Layer Outputs injections into different layers variables across 18 models with a fixed injection rate of 10%.

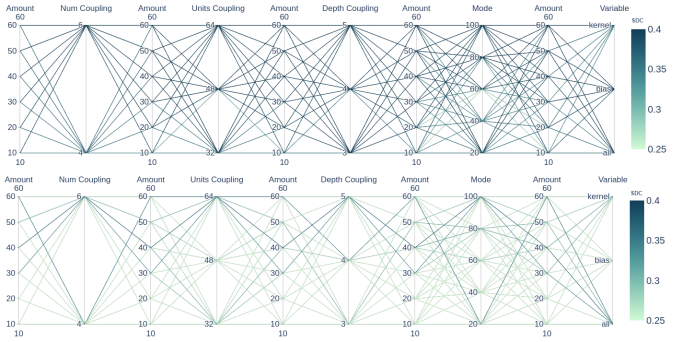


Fig. 10: Random (up) and Zeros (down) layer states experiments with all the configurations tested.

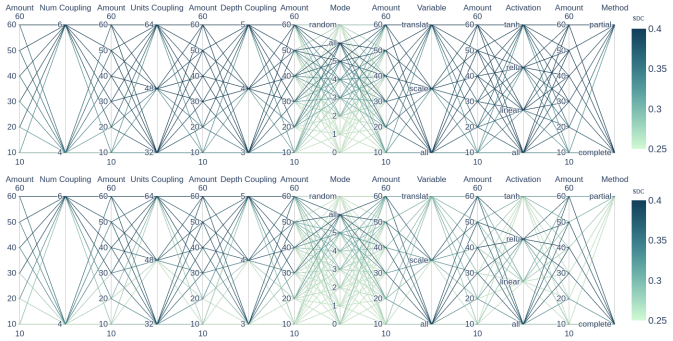


Fig. 11: Random (up) and Zeros (down) layer outputs experiments with all the configurations tested.

paring the Random and Zeros approaches, Figures 9 and 11, but distinct differences emerge depending on injection type and model configuration. For zero-value injections, lower SDC rate values are achieved with small injections. On the ”Mode” axis, higher SDC rate values are observed when injections target the last layers of the NN, with the highest SDC rate recorded when all output layers are injected. On the ”Activation” axis, randomized outputs produce consistent behaviors across different activation functions. In contrast, the Zeros approach results in lower SDC rate values for ”tanh” and ”linear” activations compared to ”relu.” From Figures 9a and 9b, models with fewer units perform worse under Zeros injection, particularly those located at the edges of the graphs. Smaller hidden layers result in information loss, making it harder for models to capture relevant features. Deeper coupling layers exacerbate this issue as errors propagate more extensively through the network. However, models with larger numbers of units mitigate these effects, retaining more information and reducing error propagation. For random-value injections (Figures 9c and 9d), performance patterns closely resemble those observed with zero-value injections but exhibit key differences. The noise introduced by random values reduces performance, as reflected by the more rounded shapes in the graphs. Despite this reduction, the separation between models remains limited, with a notable outlier being the model featuring 32 units and a depth of 5. Random noise

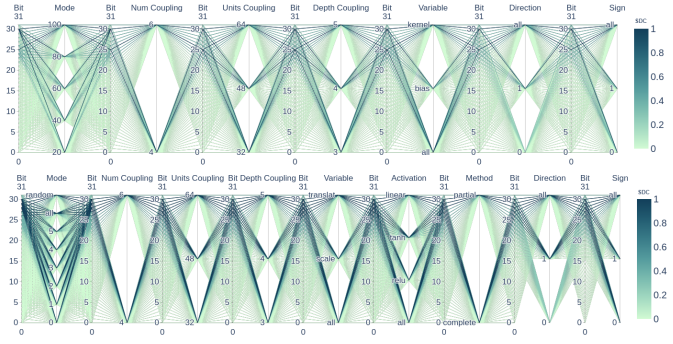


Fig. 12: Bit-flips experiments for layer states (up) and outputs (down) with all the configurations tested.

disrupts translation layers, but its effects are mitigated by scale layers, particularly in the combined ”all” configuration.

Bit-flips: As shown in Figure 12, higher exponent bits consistently result in the highest SDC rate across both configurations. For layer states bit 30 and the peak of the Gaussian at bit 25 exhibit a strong correlation with SDC rate (Figure 6a). Alterations in these positions lead to higher SDC rate values. In contrast, bit 31, associated with the sign, has a comparatively smaller impact. Layer outputs exhibit similar behavior to layer states but demonstrate poorer performance across all exponent bits, ultimately contributing to higher SDC rate values. This behavior was found in all experiments.

V. Conclusions & Future Work

This research developed a fault injection framework tailored for Real-NVP networks, designed for space applications where robustness, resilience, and compactness are critical. Our customized framework, inspired by existing solutions, like TensorFI [26], supports fault injections in two main areas: layer states, targeting network variables before inference, and layer outputs, enabling real-time injections during inference. These configurations allow for both broad testing and granular ablation studies, helping to identify the network’s strengths and vulnerabilities. The study explored three methods of fault injection: Zeros, which sets variables or outputs to zero; Random, simulating random fluctuations; and Bit-flips, mimicking bit-level alterations caused by radiation. By analyzing faults in both the variables and the outputs, we evaluated the network’s ability to classify anomalies in multivariate time-series under space-like conditions. The results offer valuable insights into improving the robustness of Real NVP networks for safety-critical applications. Future work should focus on expanding fault injection frameworks that adopt a hybrid approach, combining software/hardware-level injection methods. This will provide a more comprehensive understanding of the reliability of AI systems in challenging environments like space.

Acknowledgment

This work has been developed with the contribution of the Politecnico di Torino Interdepartmental Centre for Service Robotics (PIC4SeR https://pic4ser.polito.it).

References

[1] Johannes Benkhoff et al. "BepiColombo—Comprehensive exploration of Mercury: Mission overview and science goals". In: *Planetary and Space Science* 58 (Jan. 2010), pp. 2–20.

[2] Simone Pirrotta et al. "ArgoMoon and LICIACube: Italian first missions operated in Deep Space". In: *17th International Conference on Space Operations*. Mar. 2023.

[3] Dario Riccobono et al. "A Microsatellite-based Lunar Constellation for Communication and Navigation Services". In: *Proceedings of IAC22*. B2,4,3,x71766. 2022.

[4] Carlo Cena et al. "Physics-Informed Real NVP for Satellite Power System Fault Detection". In: *2024 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. 2024, pp. 679–684.

[5] B.E. Pritchard, G.M. Swift, and A.H. Johnston. "Radiation effects predicted, observed, and compared for spacecraft systems". In: *IEEE Radiation Effects Data Workshop*. 2002, pp. 7–13.

[6] Ramon Canal et al. "Predictive Reliability and Fault Management in Exascale Systems: State of the Art and Perspectives". In: *ACM Comput. Surv.* 53.5 (Sept. 2020).

[7] Giulio Gambardella et al. "Accelerated Radiation Test on Quantized Neural Networks trained with Fault Aware Training". In: *2022 IEEE Aerospace Conference (AERO)*. 2022, pp. 1–7.

[8] Alessandro Vallero, Dimitris Gizopoulos, and Stefano Di Carlo. "SIFI: AMD southern islands GPU microarchitectural level fault injector". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 138–144.

[9] Siva Kumar Sastry Hari et al. "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation". In: *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2017, pp. 249–258.

[10] Michael A. Kochte et al. "Efficient Simulation of Structural Faults for the Reliability Evaluation at System-Level". In: *2010 19th IEEE Asian Test Symposium*. 2010, pp. 3–8.

[11] Anderson L. Sartor, Pedro H. Becker, and Antonio C. Beck. "A fast and accurate hybrid fault injection platform for transient and permanent faults". In: *Des. Autom. Embedded Syst.* 23.1–2 (June 2019), pp. 3–19.

[12] Paolo Rech. "Artificial Neural Networks for Space and Safety-Critical Applications: Reliability Issues and Potential Solutions". In: *IEEE Transactions on Nuclear Science* 71.4 (2024), pp. 377–404.

[13] Austin P. Arechiga and Alan J. Michaels. "The Robustness of Modern Deep Learning Architectures against Single Event Upset Errors". In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. 2018, pp. 1–6.

[14] Zitao Chen et al. "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*. SC '19. Denver, Colorado: Association for Computing Machinery, 2019.

[15] Guanpeng Li et al. "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*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017.

[16] Annachiara Ruospo et al. "Investigating data representation for efficient and reliable Convolutional Neural Networks". In: *Microprocess. Microsyst.* 86.C (Oct. 2021).

[17] Majid Sabbagh et al. "Evaluating Fault Resiliency of Compressed Deep Neural Networks". In: *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. 2019, pp. 1–7.

[18] Sparsh Mittal, Himanshi Gupta, and Srishti Srivastava. "A survey on hardware security of DNN models and accelerators". In: *J. Syst. Archit.* 117 (2021), p. 102163.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[20] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 675–678.

[21] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE].

[22] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: *5th International Conference on Learning Representations (ICLR)*. OpenReview.net, Apr. 2017.

[23] Ole J. Mengshoel et al. "Diagnosing Faults in Electrical Power Systems of Spacecraft and Aircraft". In: *Proceedings of the Twenty-Third Conference on Artificial Intelligence*. Ed. by Dieter Fox and Carla P. Gomes. AAAI Press, July 2008, pp. 1699–1705.

[24] "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84.

[25] Xun Jiao et al. *PVF (Parameter Vulnerability Factor): A Scalable Metric for Understanding AI Vulnerability Against SDCs in Model Parameters*. 2024. arXiv: 2405.01741 [cs.CR].

[26] Zitao Chen et al. *TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications*. 2020. arXiv: 2004.01743 [cs.DC].