Improving Deep Neural Network Reliability via Transient-Fault-Aware Design and Training

Fernando Fernandes dos Santos, Niccolò Cavagnero, Marco Ciccone, Giuseppe Averta, Angeliki Kritikakou, Olivier Sentieys, Paolo Rech, Tatiana Tommasi

Improving Deep Neural Network Reliability via Transient-Fault-Aware Design and Training

Fernando Fernandes dos Santos, *Member, IEEE,* Niccolò Cavagnero, *Student, Member, IEEE,* Marco Ciccone, *Member, IEEE*, Giuseppe Averta *Member, IEEE*, Angeliki Kritikakou, *Member, IEEE,* Olivier Sentieys, *Senior, IEEE*, Paolo Rech, *Senior, IEEE,* and Tatiana Tommasi, *Member, IEEE,*

Abstract—Deep Neural Networks (DNNs) have revolutionized several fields, including safety- and mission-critical applications, such as autonomous driving and space exploration. However, recent studies have highlighted that transient hardware faults can corrupt the model's output, leading to high misprediction probabilities. Since traditional reliability strategies, based on modular hardware, software replications, or matrix multiplication checksum impose a high overhead, there is a pressing need for efficient and effective hardening solutions tailored for DNNs. In this paper we present several network design choices and a training procedure that increase the robustness of standard deep models and thoroughly evaluate these strategies with experimental analyses on vision classification tasks. We name *DieHardNet* the specialized DNN obtained by applying all our hardening techniques that combine knowledge from experimental hardware faults characterization and machine learning studies. We conduct extensive ablation studies to quantify the reliability gain of each hardening component in DieHardNet. We perform over 10,000 instruction-level fault injections to validate our approach and expose DieHardNet executed on GPUs to an accelerated neutron beam equivalent to more than 570,000 years of natural radiation. Our evaluation demonstrates that DieHardNet can reduce the critical error rate (i.e., errors that modify the inference) up to 100 times compared to the unprotected baseline model, without causing any increase in inference time.

Index Terms—Deep Learning, Reliability, Neutrons, GPUs, Radiation-induced faults

1 Introduction

Nowadays artificial intelligence is ubiquitous in our lives, with the number of deep network-based applications sharply increasing. A variety of novel technologies are enabled by machine learning, including image and video recognition, medical diagnosis, automatic predictive maintenance of industrial devices, and fully autonomous vehicles. While the advantages of this trend are self-evident, the potential harm resulting from adopting artificial intelligence without any reliability and security evaluation should be carefully considered. In particular, maliciously generated samples may deceive deep learning models into making incorrect predictions, and extensive research is currently dedicated to designing new algorithmic solutions to improve their robustness against multiple adversarial threats [17], [41]. Even in the most optimistic scenario of full protection from these attacks, the real world still poses several challenges to the hardware that allows deep learning to thrive. Graphics Processing Units (GPUs) with Tensor Cores for mixed precision training are essential Deep Neural Networks (DNN) accelerators that have shown faulty behavior in case of environmental disturbances, process, temperature, voltage variations, and radiation-induced soft errors [1]. The latter ones are particularly alarming since they are stochastic and transient, being very hard to detect [11], [32]. This makes them very different from permanent faults that can be detected at boot time with dedicated system checks [43]. Previous works have quantified as significant the risk that high-energy particles striking electronic devices including GPUs could cause malfunctions [30], [32]. The resulting failures lead to non-compliance with established standards. Specifically, the ISO26262 regulates functional safety of road vehicles and states that the System on Chip (SoC) responsible for DNN inferencing hardware must maintain a transient hardware fault rate below 10 Failures in Time (FIT), regardless of the DNN algorithm used and achieved accuracy. This limit corresponds to a single error in 10^9 hours of operation. Considering that the DNN hardware occupies only a fraction of the overall SoC area, the FIT allowance for the DNN accelerator should be much lower than 10 in self-driving cars. However, the radiation-induced error rate of DNN accelerators is orders of magnitude over this limit [32], underscoring the threat of transient faults.

Unfortunately, hardware protection is prohibitively expensive in terms of area, power, and design cost. Most available software solutions for enhancing reliability are either highly inefficient in terms of inference time, such as modular replication, or simply adapt classical algorithm protection strategies to DNNs, such as selective hardening [26] and checksums in convolutions [11], [25], [32]. Error Correction Code (ECC) is a widely adopted built-in solution but it only protects GPU memories rather than the functional units and may even cause an increase in the application crash rate [11], [32]. Overall, further efforts in the development of efficient and inherently robust models are needed in order to attain dependable learning systems.

In this work we integrate a set of strengthening strategies

Fernando Fernandes dos Santos, Angeliki Kritikakou, and Olivier Sentieys are with Univ Rennes, Irisa, INRIA, CNRS, France; Angeliki Kritikakou is also with Institut Universitaire de France (IUF), France
Niccolò Cavagnero, Marco Ciccone, Giuseppe Averta, and Tatiana Tommasi are with DAUIN, Polytechnic of Torino, Italy
Paolo Rech is with the University of Trento, Italy.
Manuscript received December 3, 2024

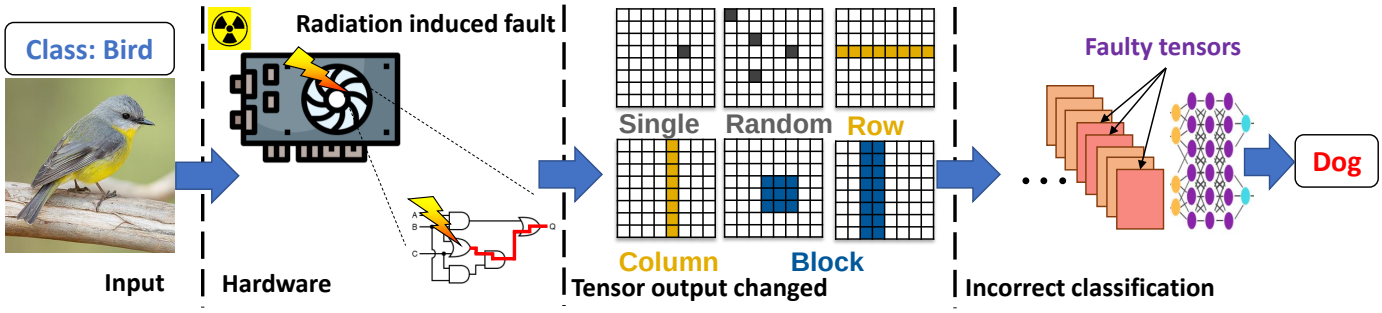


Fig. 1: Fault propagation overview. A fault can alter the operation output or memory values, modifying the DNN operation. In a feature map, errors can manifest as single or multiple corrupted values (columns, rows, or blocks of values). Incorrect tensors can ultimately lead to changes with respect to the expected prediction.

into a learning model that we name DieHardNet, a *Design ImprovEd HARDened neural Network* resilient against critical errors due to transient faults. It contains tailored network modules, designed by taking inspiration from the adversarial learning and energy-efficient literature [8], [20], [28]. Moreover, our approach benefits from an experimentally validated transient fault model on convolutional deep neural networks rather than synthetic ones used in previous publications [5], [24], [42].

We implement DieHardNet by re-engineering a ResNet [12] architecture for computer vision classification and running fault-aware training. To assess its effectiveness we conduct a three-level evaluation, including fault injections at both application and instruction levels, as well as neutron beam experiments. The obtained predictions are only marginally affected by radiation-induced soft errors with no overhead in inference time. All results confirm DieHardNet's potential, particularly on instruction-level fault simulation with an *improvement up to 106×* over a standard reference ResNet.

This paper extends our preliminary work that initially showcased the potentialities of properly designed architectural solutions and fault-aware training to improve the reliability of a deep learning model [4]. Specifically:

- we discuss the sensitivity of DNNs and thoroughly contextualize the problem in the literature, both considering fault propagation models and the limits of the existing solutions;
- we provide more details on the adopted hardening strategies that compose our DieHardNet and their effective interaction. As we realized that the occurrence of NaN values could compromise the inference process, we further introduced a NaN filter in the original model;
- we conduct an extensive experimental campaign to analyze the effectiveness of DieHardNet running on two different hardware devices under neutron beam exposure both considering ECC hardware protection enabled and disabled. The collected results, together with those of the simulation experiments with faults injected at the application level (Python code) and instruction level (microinstructions of the DNN), assert the improved reliability of our approach in realistic scenarios.

Overall DieHardNet decreases the probability of mispredictions without incurring any execution time overhead.

2 Background and Related Works

In this section, we provide a brief background on DNNs, explicitly focusing on the concepts we leverage to train and design more reliable networks. We also explore the potential impact of radiation on computing devices and how this can adversely affect the inference process of DNNs.

2.1 Deep Neural Networks

DNNs are a class of machine learning algorithms characterized by multiple interconnected modules. The network's depth refers exactly to the presence of several hidden layers between the input and output that allow the model to learn hierarchies of increasingly abstract features from the data. At the output, the loss function measures the discrepancy between the prediction and the ground truth, and its minimization guides the update of all the inner layers' parameters via backpropagation.

When dealing with visual data the DNN architectures usually include convolutional layers that capture spatially distributed structures by convolving a weight matrix (named kernel or filter) with the input. Each convolutional layer may have multiple independent kernels and get as output a corresponding number of feature maps (named channels). This operator module is usually followed by a normalization layer and an activation layer. The former stabilizes the training process by smoothing the optimization landscape [33] and preventing weight and gradient explosions. The most commonly used normalization layer is *BatchNorm*, which learns an approximation of the first and second statistical moments of each channel during training to normalize the input tensors. Instead, the activation layer introduces some non-linearity in the model and is often implemented using Rectified Linear Units, defined as $ReLU(x) = \max(0, x)$, where x is the input tensor.

The sequence of convolution-normalization-activation is usually repeated several times inside a network and the order of the layers may also change. For instance, the activation function is placed before the convolution (pre-activation) in [13], while the first operation is normalization (pre-norm) in [9]. Indeed, the layer order as well as their number and several other internal hyperparameters (e.g. dimension of the kernel, convolutional stride, and padding) are design choices that may depend on several factors including the specific task to face and possible computational constraints. All these settings are responsible for the architecture's expressivity, trainability, and reliability.

TABLE 1: Overview of the existing methods to improve DNNs reliability. Methods that rely on hardware changes add near-to-zero execution time overhead but have a high design cost. Algorithm modifications generally add overhead to inference. Existing fault-aware training are applied to quantized networks or specific accelerators. DieHardNet combines fault-aware training with DNN designs to support complex and float-based DNNs.

Method		Inference Overhead	Suitable for complex DNNs	Require HW modification
Reliable hardware and voltage scaling	[18], [24], [27], [40] [16], [29], [31]	Low	Partially (HW based).	Yes
Algorithm modification	[10], [25], [26], [32] [11], [14], [24], [27]	Medium to High	Overhead increases as the DNN complexity increases.	No
Fault aware training	[39], [44] [19], [34], [42]	Low or None	Only specific accelerators and quantized networks	No
DieHardNet	**This work**	**None**	**Yes. Proposed for complex and float based DNNs.**	**No. Transparent to the HW**

As the complexity of the DNN increases, the final model's generalization abilities are significantly influenced by both the dataset size and its variability. In this regard, data augmentation is commonly employed to create a more diverse training set encompassing valuable information not present in the original data.

2.2 Fault propagation in DNNs

Previous studies have investigated the reliability of DNNs through radiation experiments [11], [32] and fault injections [11], [24], [37]. Some works have also investigated the impact of different activations and other parameters in DNNs reliability [22]. The parallel architecture of DNN accelerators was shown to be particularly critical since the corruption of schedulers or shared resources (memories) affects multiple threads [6]. For example, a single transient fault in a GPU can corrupt the Warp Thread Scheduler, the smallest thread group in a GPU, which comprises 32 threads. This may result in the corruption of several convolution elements that are likely to propagate in the DNN, inducing mispredictions [32].

More specifically, a single transient fault spreads and corrupts multiple elements of the convolution output. This was observed in the first radiation experiments on GPUs [30] and later analyzed in detail with software and low-level fault injections [3]. Figure 1 illustrates how a fault originating from the hardware may change the result of a DNN applied for object recognition. While the GPU executes the DNN, a transient fault changes the circuit's expected value. The fault propagates through the software instructions until it reaches the feature map, i.e. the output tensor produced by a convolutional layer. The fault can corrupt the feature maps in different ways (*multiple error models*), such as single or multiple values within a tensor, an entire row or column (both often referred to as line), or a block of values. Additionally, the fault can drastically change the magnitude of the feature map values. Since the floating-point representation range is large, the corrupted values can go to *infinity* or even become *Not a Number* (NaN). These errors propagate through the network and eventually reach the last fully connected layer that produces wrong classification probabilities.

In general, we indicate as *fault model* a systematic representation of how a hardware fault manifests itself at a software visible state. What we described above are fault models obtained from physical experiments and are available open-souce to ease reproducibility and third parties analisis. It is clear that over-simplified models such as single-bit flips are marginally useful, while reliable fault models are crucial to effectively harden DNNs as they help to prepare the network for real applications.

2.3 Radiation-induced Faults and Countermeasures

Radiation can induce transient faults in hardware that propagate and manifest as different kinds of failure. The fault may (1) be *Masked* without affecting the software execution, (2) induce a *Silent Data Corruption* (SDC), which refers to incorrect application output, or (3) cause a *Detected Unrecoverable Error (DUE)*, which is a crash or a device reboot. SDCs are particularly problematic, as their silent nature makes them extremely hard to detect and can potentially lead to unstable or unknown system states. Overall, the literature agrees on some general conclusions: (1) not all SDCs are *critical* for DNNs since some errors affecting the output still allow the correct prediction; (2) the convolution tends to spread the hardware fault: a single-bit flip can corrupt a significant portion of the feature map; (3) the value of the transient corruption (i.e., how much the corrupted output differs from the correct output) is neither random nor can be simplified with a single-bit flip model.

Methods that aim at improving the reliability of DNN models by executing them on hardened hardware (such as improved memory cells and Dice flip flops) or using voltage scaling have low inference overhead since fault tolerance is fully implemented on the hardware level without performance loss [16], [18], [24], [29], [31], [40]. Although these approaches have the benefit of not requiring algorithm changes, they often come with a high implementation cost in terms of design, area, and power consumption. As DNN models grow in size and complexity, hardware-based methods may become prohibitively expensive.

Software-based hardening solutions include Algorithm-Based Fault Tolerance (ABFT) [10], [11], [25], [27], [32], and filters for identifying propagation of incorrect values [14], [24], [32]. Some interesting works have proposed solutions to mitigate the effect of permanent faults in systolic arrays or DDR faults on DNNs execution [23], [43]. While available strategies have shown some effectiveness, they only adapt solutions derived from shallow learning algorithms to DNNs, without leveraging their inherent modular structure. Furthermore, existing methods add a prohibitive overhead, as the complexity of DNNs increases since they aim at

mitigating all faults, without distinguishing between critical and tolerable SDCs: the cost of protecting DNNs from the latter is clearly unjustified.

Another viable strategy to improve DNN robustness is fault-aware training which consists in injecting noise or errors during training to simulate the effects of faults in the forward propagation [19], [39], [42], [44]. This enables the network to learn how to maintain high accuracy even in the presence of faults. Unfortunately, the majority of previous works only consider naive NN models that do not involve convolutions, or use quantized models to apply fault-aware training and require hardware modification.

Table 1 provides a summary overview of the existing fault tolerance techniques applied to computational models. None of them was originally tailored for DNNs including training with floating point operations, which are essential to get high performance in challenging tasks as those involved in computer vision applications. Moreover, they rely on synthetic fault models that might not match what is experimentally observed. As a consequence, state-of-the-art methods are either strongly dependent on the used hardware or require significant algorithm modifications, or even introduce an inference time overhead that is unbearable for key applications.

3 DieHardNet

This section describes our approach to enhance DNN reliability without requiring specialized hardware or incurring any inference time overhead. Starting from an in-depth knowledge of how faults propagate through DNN models we introduce specific network modifications that make it effective, efficient, and reliable by design. The modifications involve not only the model's architecture but also how it is trained. DineHardNet combines all the proposed strategies.

3.1 Activation Function

Most of the critical faults for DNNs (i.e., the ones that cause mispredictions) significantly modify the values propagated through the network. Besides experimental evidence [5], [32], this is intuitive as for corrupted values close to the correct ones the network's output would remain unaffected thanks to its internal approximation processes. In previous works, specific layers were introduced in the DNN to detect such high-magnitude faults with a consequent increase in computational burden and inference time [5]. We claim that adding extra layers is unnecessary since neural networks already have intrinsic tools to filter excessive values.

As explained in Section 2.1, the ReLU activation function introduces a non-linearity in the network by acting as a thresholding mechanism that allows the propagation of positive signals while not transmitting negative ones. Thus, it enforces a lower bound to the values, but it can be easily modified to include also an upper bound. Such a solution was adopted for DNNs running on mobile and embedded devices to reduce the working memory and improve quantization [21]. Indeed, in case of limited resources, the use of low-precision arithmetic can lead to faster computation but it results in numerical instability.

ReLU6 was designed to address these issues by limiting the activation output to a maximum value of six, reducing the risk of overflow/underflow. Its beneficial effect was clearly presented in [21] for various computer vision tasks such as classification, and semantic segmentation. Afterwards, by extensively tracking the statistics of the activation values, it was shown that the specific value six produced the best accuracy-reliability trade-off in the case of hardware permanent faults [14].

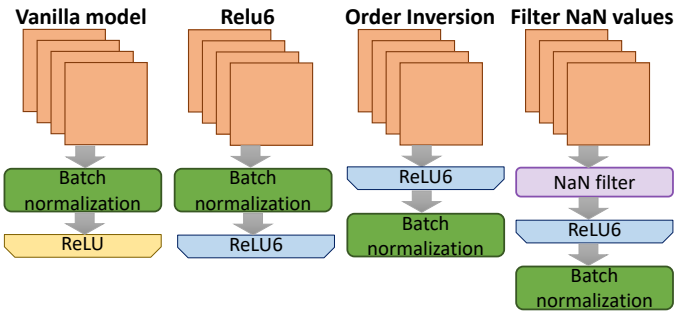


Fig. 2: Non-intrusive modifications to DNN blocks to increase reliability: ReLU6, activation layer order inversion, and filtering out *Not a Number* (NaN) values.

We remark that ReLU6, as well as other ReLU variants including an upper bound (ReLU-n, [21]), encourage the model to learn sparse features but do not reduce the overall operation precision of DNNs: they only clip values that reach the layer, allowing convolutions and weights to still have values outside of the range defined by the bounds.

Considering our scenario, even if a neutron strike perturbs a feature map by generating high-magnitude values, the ReLU6 will scale it down to the value six, reducing the impact of the error and possibly allowing the DNN to recover from this perturbation. Thus, it mitigates the effect of critical transient faults without adding extra dedicated layers or reducing the network's performance.

3.2 Network's Layer Order

In the case of errors affecting the inner layers of neural architectures, the probability of fault occurrence in the convolution operation is higher than that of the other modules, since the former is by far the most computing-expensive one. As mentioned in Section 2.1 the standard order of network layers typically involves batch normalization after the convolution. As the name says, the normalization is executed on the feature maps obtained over multiple samples in a batch, and it modifies the observed values making their scale more consistent and manageable. However, when the feature maps contain outliers due to errors or noise in the convolutional output, batch normalization may spread the problem during training, leading to an error amplification that will negatively impact the accuracy even on clean data at inference time.

From what was discussed in Section 3.1, we know that an activation function including an upper bound would help to clip the observed values and reduce the role of single outliers in the training distribution (i.e., corrupted samples). Hence, introducing such an activation directly after the convolution will significantly limit the error propagation and the following normalization layer would learn more accurate data statistics. This layer inversion has been recently

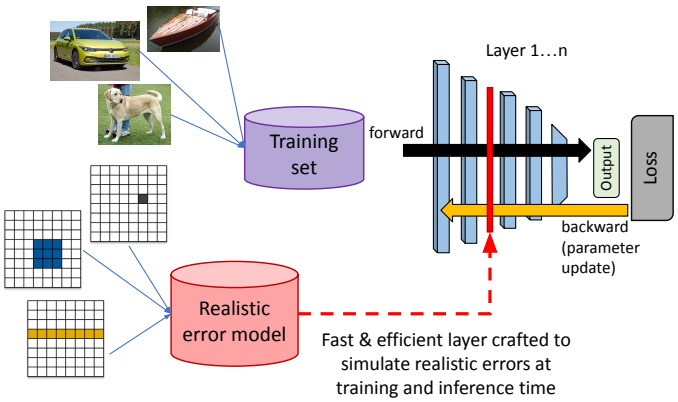


Fig. 3: Fault-aware training pipeline scheme. We randomly select the experimentally-validated fault model to apply at the output of a random convolution layer. During training, we perform ≈ 3.75 million injections.

adopted in complex architecture where network stability is crucial, even without explicit causes of faults [36].

When dealing with radiation errors, further attention should be paid to the occurrence of NaN values in the feature maps. They can compromise the inference process as even tailored activation functions such as ReLU6 would not filter them out. To overcome this issue, we set NaN values to zero. In high-level coding (such as PyTorch), NaN spotting can be easily performed through pre-defined functions or handcrafted simple routines based on checking equality conditions that require inspecting only the exponent bits of a 32-bit floating point and run in negligible time.

To summarize, an overview of the layer order that can support DNN reliability is presented in Figure 2.

3.3 Fault-Aware Training

DNNs are powerful tools however, they perform poorly when deployed in scenarios not seen during the training phase. There are several ways to address this issue [7], but data augmentation is the most common approach that does not need algorithm modifications. It provides the network with a larger variability in the training data, thus helping generalization at inference time. This strategy is also widely adopted for adversarial robustness: *adversarial training* exploits dedicated augmentations either static before the training (e.g. mixture of adversarial and clean examples) or obtained by generating samples dynamically on the basis of the model's current state [45].

The same logic of adversarial training can be adopted for transient faults. A DNN trained to get high accuracy while observing transient faults would produce a robust model with predictions less affected by errors at inference time as the network familiarized with the occurrence of neutron-induced errors. In particular, as shown in Figure 3, while performing the forward pass of the DNN training, the feature maps are randomly corrupted (convolution output) in a given layer of the network, injecting errors according to an experimentally observed fault model. As a result, the network adjusts the learned weights to reduce the likelihood of a misprediction.

The effect of this process depends on the amount and type of faults injected into the network during the training phase. A high number of faults might prevent training convergence, while a low number might be ineffective. In terms of types, random errors do not necessarily affect all the layers as the model may easily learn how to deal with them. For instance, DNN training tends to be transparent to single-bit flips. Clearly, knowing the fault model, as we do thanks to our experimental characterization, provides a relevant contribution to the design of the fault-aware training process.

4 Fault Injection Methodology through Simulation or Radiation Exposure

Studying hardened DNNs' capabilities requires analyzing their behavior under transient fault conditions. Exposing the whole GPU to radiation via a *neutron beam* provides a real testbed, however these experiments are highly costly, time consuming, and do not allow for tracking fault propagation.

A possible alternative consists of targeting specific components of the processing unit via software fault simulations. We are aware that simplistic random single bit-flip fault injectors lead to unreliable evaluations [38]. Thus, we leverage previous studies that characterized how the physical fault manifests at the software visible state with models that have been already validated to be sufficiently accurate for ARM devices [2] and GPUs [32]. Specifically, it is possible to inject faults at the *application* level (Python code) or at the *instruction* level (microinstructions of the DNN). In the first case, the errors are crafted to appear directly after specific network layers. In contrast, in the second case, the errors propagate seamlessly from machine instructions to the network output.

Overall, we consider fault-injection methodologies at the physical as well as two different software levels, to fully validate our mitigation solution.

4.1 Application-level Fault Injections

As discussed in Section 2.2, empirical observations show that transient hardware faults manifest as errors in the output of convolutional layers [3], [25], [32], with different fault models describing errors as single, lines, or blocks of wrong values in the produced feature maps. To create this effect we elaborated a tailored error injector module (Figure 3) that selects the fault model and multiplies the output tensor by a sampled uniform random value that sets the error magnitude.

The process can be directly implemented on GPUs by applying a mask to a tensor. It can be easily integrated into any network learning phase (i.e., fault-aware training), allowing the injection of even millions of faults with a negligible training time overhead. Specifically, for ResNet44 on CIFAR10 a clean training epoch takes 9.7s while the fault-injected one takes 9.8s. For ResNet56 on TinyImagenet the respective timings are 50.3s vs 50.7s.

In terms of validation, the network performance is assessed by comparing the accuracy in case of fault-free and faulty executions. We call their difference *Regret*, with low values indicating high reliability.

4.2 Instruction-level Fault Injections

To perform an injection campaign at the instruction level we exploited NVBitFI [37], an existing tool able to manage dynamic libraries and easily apply transient fault models that affect a single-thread or multiple-threads.

By following experimental evidence from the literature [3], [6], for the single-thread injections we used Single Bit Flip, Random, and Zero Values on a single floating-point instruction. For the multiple-thread injections, we selected all the threads within a GPU warp - the smallest thread execution group on a GPU - and replaced the output of a floating-point instruction with a random or zero value. To obtain realistic results, we employed a combination of fault models for instruction-level fault injections (2,000 injections per configuration). By corrupting also multiple threads in a single injection we are modeling in software the manifestation of a single hardware fault in a critical or shared resource. This fault model has been cross validated with beam experiments

Such instruction level injections on large DNNs impose a high overhead, taking up to several minutes for a single injection. As a result, utilizing this approach for fault-aware training becomes impractical. However, evaluating a hardened network's performance under such faults remains valuable for gauging its reliability.

The probability for an injected fault to be propagated from the assembly instruction to the application output is indicated as Program Vulnerability Factor (PVF, [35]). Lower values signify enhanced robustness.

4.3 Neutron beam Exposure

We ran extensive neutron beam experiments at the ChipIR facility in Didcot, UK. It delivers a neutron energy spectrum similar to the terrestrial spectrum, albeit with an accelerated flux. More precisely, at ChipIR, the neutron flux ranges between $1\times10^5 n/(cm^2\times s)$ and $2.5\times10^6 n/(cm^2\times s)$, which is approximately six to eight orders of magnitude higher than the atmospheric neutron flux at sea level ($13n/(cm^2\times h)$). We performed analyses for a total of 50 hours of beam time: it translates to at least 50×10^8 hours of natural operation when scaled to the terrestrial environment, which is equivalent to approximately 570,000 years of operation.

To simulate realistic conditions we chose a beam spot with a diameter of 2cm, which is sufficient to irradiate the whole GPU core without hitting the onboard DRAM. The device being tested was connected to a server via Ethernet. The main server could start and stop programs and reboot the device automatically. We included software and hardware watchdogs to detect application crashes or system hangs (Figure 4). The software watchdog monitored the application under study, and if it failed to respond within a predetermined time interval, the kernel was killed and restarted. This watchdog was designed to detect kernel crashes or software hangs, such as application crashes or control flow errors that prevent the GPU from completing its assigned tasks (for example, infinite loops). The hardware watchdog was an Ethernet-controlled switch that triggered a power cycle of the host computer if it did not acknowledge ping requests within a specified time frame. This watchdog is essential for detecting operating system hangs.

To ease reproducibility, all the source codes, software tools, and referenced artifacts used for our analysis are accessible in online repositories [1] [2].

The neutron beam allows us to estimate the error rates in the application, which are measured in terms of Failure In Time (FIT). This metric indicates the estimated number of neutron-induced failures expected to occur in a billion hours of operation (the lower the better). It is essential to highlight that the parallel utilization of hardware resources directly influences the FIT rate. In other words, the FIT rate increases when more resources are used in parallel. In our study, we meticulously evaluate the impact of each design approach to ensure fair FIT rate comparisons.

5 Experimental analysis

The model design and learning approach that define DieHardNet, as described in Section 3, are strategies applicable to any network and hardware architecture. The same holds for the evaluation methodology detailed in Section 4.

For our experimental analysis we decided to focus on computer vision as it is one of the areas that is majorly moving the progress of artificial intelligence and for which trustworthy models would have a large impact in sensitive applications (e.g. autonomous driving).

Specifically, we consider object classification and the obtained results demonstrate the effectiveness of DieHardNet in improving reliability without accuracy degradation or inference time overhead.

5.1 Setting

We selected a standard ResNet [12] as a case study and we trained the network from scratch on three common object classification benchmarks, CIFAR10, CIFAR100, and TinyImageNet. The first two contain 50,000 32×32 images for training and 10,000 for testing, with 10 and 100 different classes, respectively. TinyImagenet contains 100,000 64×64 images for training and 10,000 for testing, with 200 classes. We employ ResNet44 for CIFAR datasets and ResNet56 for TinyImageNet.

We conducted an incremental study to detail the impact of each of the DieHardNet components, starting from the use of ReLU6 and then introducing either fault-aware training (shortened as *train aware*) or inverting the layer order (shortened as *order inversion*). Finally, the whole DieHardnet includes all these strategies together with the NaN filter. To summarize, we assess the performance of five DieHardNet sub-configurations considering the corresponding naïve ResNet as baseline: (1) +ReLU6, (2) +ReLU6+Train aware, (3) +ReLU6+Order inversion, (4) +ReLU6+Train Aware+Order Inversion, and (5) +ReLU6+Train Aware+Order Inversion+NaN filter.

5.2 Training Details

Since our goal is to mitigate the effect of the neutron-induced faults, rather than achieving the maximum possible test accuracy, we opt for a simplified training hyperparameter setting. In particular, we trained for 100 epochs with

1. DieHardNet code: github.com/diehardnet/diehardnet.
2. Experimental setup: github.com/diehardnet/diehardnetradsetup.

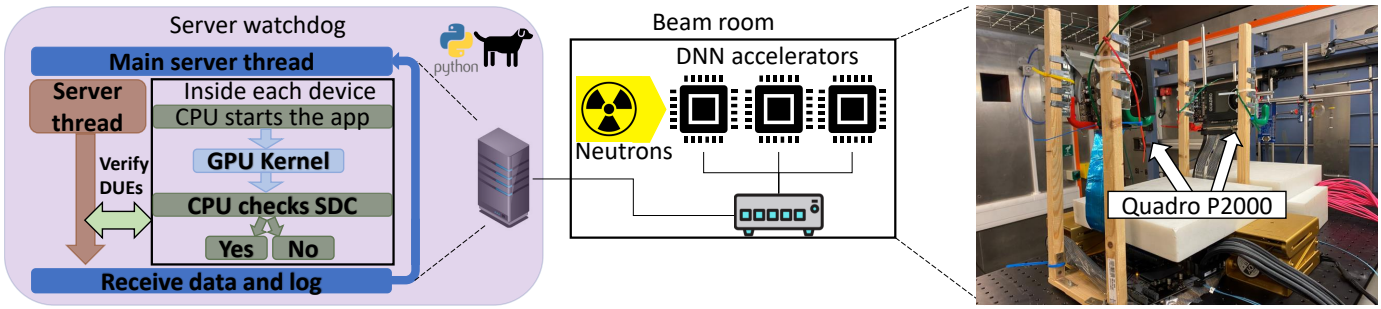


Fig. 4: The DNN accelerators are exposed to a neutron flux and connected through Ethernet. A host platform (outside the beam spot) is responsible for managing communication between the accelerators and the server. A software watchdog server monitors the events on the devices under test. In case a device stops responding, the server can automatically perform a power cycle or reboot the operating system.

random crop and horizontal flipping as data augmentations. We employed a standard Stochastic Gradient Descentoptimizer, a Cosine Annealingscheduler, and a Binary Cross Entropy loss. The initial learning rate was set to 2. We used a batch size of 128, weight decay of 1e-5, and gradient clipping of 1 to prevent exploding gradients during training. We used Titan V GPUs as training device. We remark that with this configuration DieHardNet shows a comparable accuracy with respect to a non-hardened ResNet in the fault-free scenario, i.e. 92.23% accuracy for ResNet44 and 91.31% accuracy for DieHardNet on CIFAR10 dataset.

When performing fault-aware training (faults introduced at the application level), we select the convolution layer affected by the injection, and sample 75% of the training images in the forward pass to which the error models are applied. Indeed a preliminary analysis revealed that injecting faults in every sample prevented convergence, and $3/4$ was the highest feasible fraction of faulty instances in a batch.

Moreover, through our fault injector, we linearly increase the error magnitude in a curriculum-learning fashion while the epochs advance: this favors the model in learning to reject errors with growing severity.

We observed that training by injecting single value errors did not increase the model reliability (the network internal weights remain unaltered), while models trained with block errors alone achieved the same reliability as those trained with all error models (jointly single, line, and block). Therefore, fault-aware training is performed by exclusively using block error injections to speed up the process.

5.3 Results - Application-level Fault Injections

Figure 5 shows the Regret (vertical axis) for each DieHardNet configuration (horizontal axis) when injecting application-level faults. As NaN values are not observed at this fault injection level, applying the NaN filter does not change the results, thus the corresponding column is discarded in the bar plot.

On average, the baseline ResNet44 exhibits a drop in accuracy of over 13% on all datasets. By replacing the ReLU activation with ReLU6, we observe a significant improvement in protection, with a $\approx 8\%$, 3% and 5% increase in accuracy on CIFAR10, CIFAR100 and TinyImageNet respectively. This result supports our hypothesis that ReLU6 can limit the magnitude and propagation of errors in the architecture and can thus play a crucial role in designing robust DNNs.

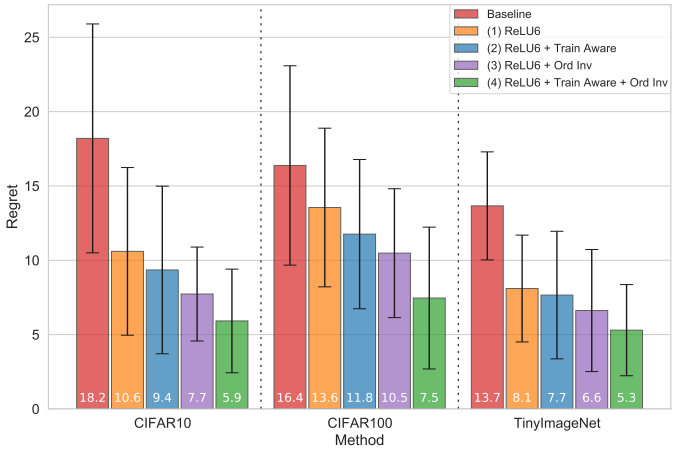


Fig. 5: Application-level Fault Injection results: average regret of different DieHardNet configurations on CIFAR10 (left), CIFAR100 (center), and TinyImageNet (right). The error bars are generated by averaging across 5 different injection campaigns. Standard deviation across the campaigns is reported. As NaN values are not observed with this level of fault injections, applying the NaN filter does not change the results.

Interestingly, the improvement provided by adding fault-aware training is quite limited. We attribute this low effectiveness to poorly learned statistics in the normalization layers due to the perturbations induced by noise injections during training. This behavior is closely related to the model architecture, which puts normalization before activation. In other words, in the vanilla design, the normalization acts on non-clipped features, leading to improper learned statistics and affine transform (details in Section 3.2).

Introducing the layer order inversion appears slightly more convenient than using ReLU6 with fault-aware training, while a major gain is obtained when all the strategies are combined together in DieHardNet with an increase in test accuracy of up to 12.3%, 8.9% and 8.4% on CIFAR10, CIFAR100, and TinyImageNet respectively.

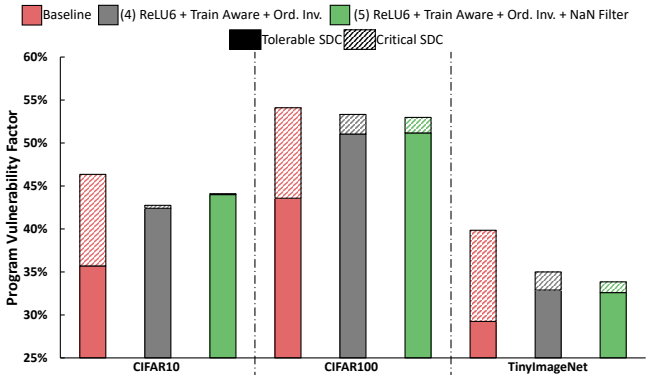


Fig. 6: Program Vulnerability Factor (PVF) for CIFAR10, CIFAR100, and TinyImageNet datasets. The PVF is divided into tolerable and critical SDCs. Since our injections are limited to the output of floating-point instructions, no DUEs are observed. Note that the y-axis goes from 25% to 60%.

5.4 Results - Instruction-level Fault Injections

Differently from what happens with application-level fault injections, when introducing instruction-level faults they do not always manifest explicitly. Thus, to assess the behavior of a hardened network it is important to monitor SDCs, specifically distinguishing between *Tolerable* and *Critical* ones with only the latter resulting in changes to the expected inference (misprediction).

Figure 6 presents the PVF of the DieHardNet sub-configurations for CIFAR10, CIFAR100, and TinyImageNet datasets. As configurations (1), (2), and (3) did not provide any benefit in terms of Regret reduction on the application-level fault injection, we focus our analysis on the best configurations (4) and (5).

Interestingly, for the simpler datasets (CIFAR10 and CIFAR100), the total number of SDCs (Tolerable+Critical) is similar between the Baseline and the hardened DNNs: 46.35% for the Baseline, 42.75% for configuration (4), and 44.10% for configuration (5), on CIFAR10 dataset; 54.10% for the Baseline, 53.33% for configuration (4), and 52.98% for configuration (5), on CIFAR100 dataset. This suggests that the fault injection impacts the DNN execution, and failures manifest in the output in both cases. However, the hardened DNN can still provide the expected prediction (as in the fault-free scenario) even in the presence of faults. Specifically, DieHardNet with all its internal hardening strategies presents only 0.10% Critical SDC PVF for CIFAR10 and 1.80% for CIFAR100, whereas the corresponding values for the Baseline models are 10.65% and 10.53%, respectively. Such an advantage with a reduction of up to two orders of magnitude in PVF clearly indicates the remarkable robustness of DieHardNet.

We can also draw similar conclusions from the TinyImageNet results. The Overall SDC PVF decreases when passing from the Baseline (39.85%) to configuration (4) (35.00%) and (5) (33.85%), confirming the beneficial effects of DieHardNet. The Critical SDC PVF presents the same trend observed in the case of the CIFAR datasets with a value of 10.60% for the Baseline and 2.10%, 1.25%, respectively, for configurations (4) and (5).

5.5 Results - Neutron beam Exposure

Besides the simulation experiments described above, we performed an extensive analysis of our hardened classification model running on two different hardware devices under neutron beam exposure. Considering the results already discussed for the intermediate network configurations and the beam time limitations, we tested only the whole DieHardNet including all its hardening components: +ReLU6+Train aware+Order Inversion+NaN filter on the CIFAR100 dataset.

The two devices are the NVIDIA Pascal (Quadro P2000) and the Ampere (RTX A2000) GPUs. The former is based on the Pascal Instruction Set Architecture (ISA) and fabricated in TSMC FinFET technology with a 16nm process node. The latter is based on the Ampere microarchitecture and is also built with Samsung's 8nm 8N NVIDIA Custom Process technology. Only the RTX A2000 microarchitecture provides Single Error Correction Double Error Detection (SECDED) Error Correcting Code (ECC) to protect the register file, shared memory, caches, and memory buffers. As part of our validation, we assess the effectiveness of DieHardNet when ECC is enabled and disabled.

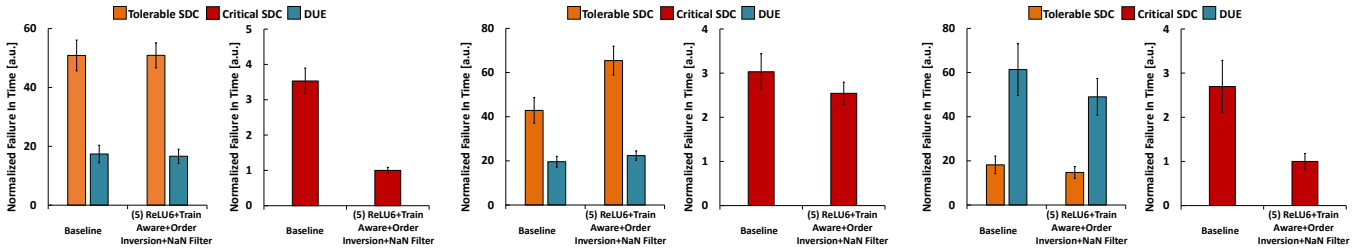
Figure 7a displays the Failure In Time (FIT) rates obtained from beam experiments on the Pascal GPU, comparing the Baseline with DieHardNet. As system crashes and device rebooting may naturally occur in this setting, we present the Detected Unrecoverable Errors (DUE) together with the Tolerable SDC. The Critical SDC are less likely, as expected according to previous studies [24], [32], and are shown in a separate bar plot to ease visualization.

Since the FIT rate is business-sensitive information, the presented results have been normalized by the lowest Critical SDC rate (obtained by DieHardNet). The normalization is performed per board and it enables a straightforward comparison between different network configurations within the same GPU. Values are reported with 95% confidence intervals considering a Poisson distribution.

For Pascal GPU, the Tolerable SDC rate is 50.88, and the Critical SDC rate is 3.53. Even if the Critical SDC rate of the Baseline is relatively low, we know that a reliable model should completely avoid those errors. DieHardNet shows a similar Tolerable SDC rate as the Baseline (50.91) with a reduced critical SDC FIT rate (1.00). This is an advantage of 71.67% less Critical SDCs compared to the Baseline. We highlight that the network design of DieHardNet is different from that of the Baseline, but the amount of operations is almost the same with the Tolerable FIT rate that remains essentially unchanged in the two cases, and negligible variations in the DUE rate. Both networks saturate the hardware resources on Pascal GPUs: for DieHardNet the added NaN filters can be only executed sequentially.

Figure 7b shows the results on the Ampere GPU, with ECC disabled, and Figure 7c shows the results on Ampere GPU with ECC enabled.

Without ECC, the DieHardNet Tolerable SDC rate is 34.52% higher than that the Baseline. This behavior can be explained considering that the NaN filter kernels increase the register file usage by 16.40% (requiring 14 registers per thread to perform the filter GPU kernel). Additionally, the Ampere GPU resources are not saturated (differently from



(a) Pascal GPU ECC Disabled (b) Ampere GPU ECC Disabled (c) Pascal GPU ECC Enabled

Fig. 7: Pascal and Ampere GPUs' Normalized Tolerable SDC, Critical SDC, and DUE Failure In Time (FIT) rates for the Baseline and DieHardNet (configuration 5). The rates are normalized using the DieHardNet Critical SDC FIT (i.e. the lowest FIT Rate). For Ampere, the normalization utilizes the FIT rate when ECC is enabled.

the Pascal case), thus the NaN filter kernel operations are executed in parallel. As the registers used for hardening are not protected, it is natural to expect a higher SDC rate for DieHardNet since the FIT rate is linearly dependent on the number of used resources (see also Section 4.3).

By focusing on the Critical SDC we see that the Baseline shows a 1.19× higher FIT rate than DieHardNet, which confirms the advantage of our hardening technique.

Enabling ECC reduces the amount of experienced Tolerable SDC errors. Compared to the case with ECC disabled, the Tolerable SDC FIT rate decreased by 57.51% for the Baseline and 77.45% for DieHardNet. Interestingly, ECC is less effective in protecting DNNs than for typical HPC applications, for which ECC reduces the SDC rate by one order of magnitude This is because memory errors may be less problematic for DNNs than for HPC codes. Furthermore, ECC is indiscriminately applied to all memory values, which may also mask errors that would not generate a critical SDC.

In terms of Critical SDC, with ECC enabled, DieHardNet achieves the smallest FIT rate on the Ampere GPU (1.00 Critical SDC FIT, 93.22% smaller than the Tolerable SDC FIT). Additionally, the Baseline has a Critical SDC FIT that is 2.7× higher than the DieHardNet, a value that asserts the effectiveness of the proposed fault tolerance strategies in realistic scenarios.

Finally, some attention should be dedicated to the unrecoverable errors. Indeed enabling ECC leads to a potentially alarming 3.13× increase in the DUE rate for the Baseline and a 2.19× increase for DieHardNet. Previous studies [15] have established that DUEs are not caused by faults in the functional units. Instead, they may arise from device-host synchronization, illegal or misaligned memory access, and fault-induced deadlocks.Although ECC can correct single-bit flips and detect double-bit flips, it triggers a system exception upon detecting a double-bit flip, which can result in application crashes.

We further investigated the causes of DUEs on GPUs when ECC is both disabled and enabled. In the first case, DUEs primarily arise from launch errors, illegal or misaligned memory addresses, illegal instructions, and system crashes caused by the GPU driver. Unprotected caches, register files, and shared memories can directly lead to illegal memory access or corruption of instruction opcodes, thereby contributing to DUEs. In contrast, with ECC enabled, most DUEs (73.37% on average) come from exceptions generated by the ECC double error detection mechanism.

6 Discussion

DieHardNet leverages the intrinsic learning potential of DNNs and significantly reduces the Critical SDC rate while maintaining the prediction accuracy and inference time unaltered with respect to the related Baseline without protection strategies. What we have proposed is an algorithmic-based methodology that can be easily implemented on complex DNN architectures without requiring hardware modifications. Our experimental evaluation demonstrated that DieHardNet can significantly reduce the impact of faults in the model output (up to 2 orders of magnitude) without affecting or potentially reducing the DUE rate.

Table 2 compares DieHardNet with state-of-the-art reliability improvement techniques for DNNs.

DieHardNet provides a significant improvement over previous solutions that rely on hardware modification, such as FAT [42], ABAEC [40], and DARA [16]. These approaches share with DieHardNet the benefit of not adding overhead to the execution time, but they need customized hardware which introduces implementation complexity and extra costs. For example, the DARA approach requires MLC STT-RAM to be used in the accelerator [16], making it unfeasible to be implemented on commercial accelerators like GPUs. Similarly, FAT utilizes hardware Triple Modular Redundancy (TMR) and fault-aware training. This results in certain circuit parts having a 3× additional non-negligible area overhead.

We remark that the Fault Aware Training that gives the name to the FAT methods [19], [42] differs from that used by DieHardNet. Those approaches were designed for naive networks (some quantized to 8-bit integers) and do not apply to complex floating-point DNNs. They have not been validated with beam experiments and do not consider the occurrence of NaN or infinity values.

Algorithm-Based Fault Tolerance (ABFT) [32], and Algorithm-Based Error Detection (ABED) [11] applied at the software level can achieve detection rates as high as 98.96%. However, they still have a significant execution time overhead (57.00% for the ABFT and 23.00% for ABED).

Other software fault tolerance methods such as SmartPooling [32] and Ranger Restrict [5] apply value restriction at the software level (i.e., the float point values are restricted to an acceptable range of values). The achieved protection is limited since the Critical SDC rate can be high (SmartPooling covers only 85.90% of the Critical SDCs) when evaluated on neutron beam experiments. It is worth

TABLE 2: Comparison of DieHardNet with the state-of-the-art DNN hardening techniques. DieHardNet is the only technique not requiring hardware modifications, applies to complex GPUs, and is validated through beam experiments. DieHardNet reliability is comparable to or better than existing strategies, considering that some (Ranger) were validated using a naive fault model (single-bit flip).

	Overhead		Beam	Critical SDC
	Time	Hardware	Val.	Coverage
FAT [42]	0.00%	Partial TMR	No	98.49%
SmartPooling [32]	0.10%	None	Yes	85.90%
Ranger [5]	0.50%	None	No	98.96%
ABFT [32]	57.00%	None	Yes	60.00%
ABAEC [40]	0.20%	59.00%	No	99.00%
ABED [11]	23.00%	None	Yes	87.50%
DARA [16]	0.00%	STT-RAM	No	98.20%
DieHardNet	**0.00%**	None	Yes	93.22%

noting that Range Restrict has only been validated using fault simulation limited to single-bit flip, which is why the Critical SDC rate is so low.

The strategies at the basis of DieHardNet can be easily applied to various deep architectures (e.g. Transformers [9]) and when solving different tasks (e.g. object segmentation). However, they require a network to be trained from scratch and the learning process may take a long time. One way to avoid it consists of defining new hardening methodologies that can be applied to state-of-the-art pre-trained models so that a short fine-tuning phase would be enough to obtain reliable predictions for any new application. We will investigate this transfer learning direction in future work.

Moreover, we plan to perform beam experiments on various DNN accelerators so to formalize a fault model that can be used for various devices. This will further encourage hardware generalization in fault-aware training.

7 Conclusions

In this paper, we have leveraged knowledge about the internal functioning of DNNs as well as about radiation-induced error generation and propagation to design a deep classification model with improved reliability against transient faults. We proposed four hardening strategies, including the use of a clipped activation, fault-aware training, layers reordering, and NaN values filtering.

Through application and instruction-level fault injections and beam experiments, we have shown that our hardening strategies can significantly reduce the occurrence of mispredictions by up to two orders of magnitude, without imposing any overhead on the inference time. Our findings pave the way to new deep learning architecture reliable by design.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 899546, with the support of the Brittany Region. It was partially funded by ANR FASY (ANR-21-CE25-0008-01) and ANR RE-TRUSTING (ANR-21-CE24-0015-02). This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 101008126 (RADNEXT). ChipIR provided and supported neutron beam time experiments (DOI 10.5286/ISIS.E.RB2200303). We acknowledge Dr. Christopher Frost, Dr. Maria Kastriotou, and Dr. Carlo Cazzaniga for their help with beam experiments. This study was also carried out within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013).

References

[1] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[2] P. R. Bodmann, G. Papadimitriou, R. L. R. Junior, D. Gizopoulos, and P. Rech, "Soft error effects on arm microprocessors: Early estimations versus chip measurements," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2358–2369, 2022.

[3] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 984–997, 2023.

[4] N. Cavagnero, F. D. Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Transient-fault-aware design and training to enhance dnns reliability with zero-overhead," in *IEEE IOLTS*, 2022.

[5] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *DSN*, 2021.

[6] J. E. R. Condia, B. Du, M. Sonza Reorda, and L. Sterpone, "Flexgripplus: An improved gpgpu model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[7] G. Csurka, T. Hospedales, M. Salzmann, and T. Tommasi, *Visual Domain Adaptation in the Deep Learning Era*. Springer International Publishing, 2022.

[8] J. Deng, Y. Fang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu, "Retraining-based timing error mitigation for hardware neural networks," in *DATE*, 2015.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.

[10] D. Filippas, N. Margomenos, N. Mitianoudis, C. Nicopoulos, and G. Dimitrakopoulos, "Low-cost online convolution checksum checker," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 2, pp. 201–212, 2022.

[11] S. K. S. Hari, M. Sullivan, T. Tsai, and S. W. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2021.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016.

[14] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *DATE*, 2020.

[15] K. Ito, Y. Zhang, H. Itsuji, T. Uezono, T. Toba, and M. Hashimoto, "Analyzing due errors on gpus with neutron irradiation test and fault injection to control flow," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1668–1674, 2021.

[16] M. Jasemi, S. Hessabi, and N. Bagherzadeh, "Enhancing reliability of emerging memory technology for machine learning accelerators," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 2234–2240, 2021.

[17] S. Y. Khamaiseh, D. Bagagem, A. Al-Alaj, M. Mancino, and H. W. Alomari, "Adversarial deep learning: A survey on adversarial attacks and defense mechanisms on image classification," *IEEE Access*, vol. 10, pp. 102 266–102 291, 2022.

[18] J.-S. Kim and J.-S. Yang, "Dris-3: Deep neural network reliability improvement scheme in 3d die-stacked memory based on fault analysis," in *DAC*, 2019.

[19] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. Sathe, "MATIC: Learning around errors for efficient low-voltage neural network accelerators," in *DATE*, 2018.

[20] S. Koppula, L. Orosa, A. G. Yağlıkçı, R. Azizi, T. Shahroodi, K. Kanellopoulos, and O. Mutlu, "Eden: Enabling energy-efficient, high-performance deep neural network inference using approximate dram," in *MICRO*, 2019.
[21] A. Krizhevsky, "Convolutional deep belief networks on cifar-10," http://www.cs.utoronto.ca/ kriz/conv-cifar10-aug2010.pdf, 2010.
[22] S. Kundu, S. Banerjee, A. Raha, S. Natarajan, and K. Basu, "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 485–498, 2021.
[23] S. Kundu, A. Raha, S. Banerjee, S. Natarajan, and K. Basu, "Analysis and mitigation of dram faults in sparse-dnn accelerators," *IEEE Design & Test*, vol. 40, no. 2, pp. 90–99, 2023.
[24] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC*, 2017.
[25] F. Libano, P. Rech, and J. Brunhaver, "Efficient error detection for matrix multiplication with systolic arrays on fpgas," *IEEE Transactions on Computers*, pp. 1–14, 2023.
[26] A. Mahmoud, S. K. Sastry Hari, C. W. Fletcher, S. V. Adve, C. Sakr, N. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing selective protection for cnn resilience," in *ISSRE*, 2021.
[27] T. Marty, T. Yuki, and S. Derrien, "Enabling overclocking through algorithm-level error detection," in *FPT*, 2018.
[28] D. Mishkin, "Evaluation of batchnorm layer performance on imagenet-2012," 2016. [Online]. Available: https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md
[29] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ISCA*, 2016.
[30] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on gpus," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2797–2804, 2013.
[31] B. Salami, O. S. Unsal, and A. C. Kestelman, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," in *SBAC-PAD*, 2018.
[32] F. F. d. Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.
[33] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *NeurIPS*, 2018.
[34] A. Siddique and K. A. Hoque, "Exposing reliability degradation and mitigation in approximate DNNs under permanent faults," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 555–566, 2023.
[35] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *HPCA*, 2009.
[36] A. Trockman and J. Z. Kolter, "Patches are all you need?" *Transactions on Machine Learning Research*, 2023.
[37] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *DSN*, 2021.
[38] A. Tyagi, Y. Gan, S. Liu, B. Yu, P. Whatmough, and Y. Zhu, "Thales: Formulating and estimating architectural vulnerability factors for dnn accelerators," in *HPCA*, 2023.
[39] D. Xu, K. Xing, C. Liu, Y. Wang, Y. Dai, L. Cheng, H. Li, and L. Zhang, "Resilient neural network training for accelerators with computing errors," in *ASAP*, 2019.
[40] Z. Xu and J. Abraham, "Safety design of a convolutional neural network accelerator with error localization and correction," in *ITC*, 2019.
[41] M. Xue, C. Yuan, C. He, Y. Wu, Z. Wu, Y. Zhang, Z. Liu, and W. Liu, "Use the spear as a shield: An adversarial example based privacy-preserving technique against membership inference attacks," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 153–169, 2023.
[42] U. Zahid, G. Gambardella, N. J. Fraser, M. Blott, and K. A. Vissers, "FAT: training neural networks for reliable inference under hardware faults," in *ITC*, 2020.
[43] J. J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *VTS*, 2018.
[44] L. Zhao, Y. Zhang, and J. Yang, "Aep: An error-bearing neural network accelerator for energy efficiency and model protection," in *ICCAD*, 2017.
[45] W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Adversarial training methods for deep learning: A systematic review," *Algorithms*, vol. 15, no. 8, 2022.

Fernando Fernandes dos Santos received his master's and Ph.D. degrees from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2017 and 2021, respectively. He specializes in the areas of parallel architecture reliability, efficient hardening techniques, and radiation-induced effects mitigation. Currently, he is working at INRIA Rennes on improving the fault tolerance and reliability of RISCV and Deep Neural Networks accelerators.

Niccolò Cavagnero is an ELLIS Ph.D. student in Computer Vision at the ELLIS unit of Polytechnic of Turin, where he received his B.S. in Management Engineering and his M.S. in Data Science and Engineering. Niccoló research focuses on Neural Architecture Search and on the design of efficient Deep Neural Networks.

Marco Ciccone is an ELLIS Postdoctoral Researcher in the VANDAL group at Polytechnic of Turin. He received a Ph.D. in Computer Science and Engineering cum laude at Polytechnic of Milan, working on iterative and conditional models for visual representation learning. His research interests are in the intersection of meta, continual, and federated learning to scale the training of agents with heterogeneous data and mitigate the effect of catastrophic forgetting and heterogeneity across tasks, domains, and devices.

Giuseppe Averta is currently Assistant Professor of Robotics and Machine Learning at the Polytechnic of Turin. He got his PhD in Robotics and Information Engineering cum laude from the University of Pisa in 2020. His work was awarded with the Georges Giralt Ph.D. Award 2021 (for the best European PhD thesis). He is also an Italian Institute of Technology Alumnus. Giuseppe's research interests are focused on the optimization of machine learning models for edge computing, deep learning for human action recognition, and on human-inspired design, planning, and control guidelines for autonomous, collaborative, assistive, and prosthetic robots.

Angeliki Kritikakou is an Associate Professor at Univ. Rennes and IRISA/INRIA Rennes research center. She received her Ph.D. in 2013 from Univ. Patras, Greece, in collaboration with IMEC Center, Belgium. Her research interests include embedded systems, real-time systems, hardware/software co-design, mapping methodologies, design space exploration methodologies, memory management methodologies, low power design and fault tolerance.

Olivier Sentieys is a Professor at the University of Rennes holding the Inria Research Chair on Energy-Efficient Computing Systems. He is leading the Taran team common to Inria and IRISA Laboratory. His research interests are in the area of computer architectures, computer arithmetic, and embedded systems, with a focus on system-level design, energy-efficient hardware accelerators, approximate computing, fault tolerance, and energy harvesting sensor networks.

Paolo Rech received his master and Ph.D. degrees from Padova University, Padova, Italy, in 2006 and 2009, respectively. Since 2022 Paolo is an associate professor at Università di Trento, in Italy and since 2012 he is an associate professor at UFRGS in Brazil. He is the 2019 Rosen Scholar Fellow at the Los Alamos National Laboratory, he received the 2020 impact in society award from the Rutherford Appleton Laboratory, UK. In 2020 Paolo was awarded the Marie Curie Fellowship. His main research interests include the evaluation and mitigation of radiation-induced effects in autonomous vehicles for automotive applications and space exploration, in large-scale HPC centers, and quantum computers.

Tatiana Tommasi is Associate Professor in the department of Control and Computer Engineering of Polytechnic of Turin (IT), Affiliated Researcher at the Italian Institute of Technology, and director of the ELLIS Unit in Turin. She received the Ph.D. at EPFL Lausanne (CH) in 2013 and has published more than 50 papers at top conferences and journals in machine learning and computer vision. She has a strong record in theoretically grounded algorithms for automatic learning from images with robotics, medical and human-machine interaction applications. She pioneered the area of transfer learning in computer vision and has extensive experience in domain adaptation, generalization, multimodal and open-set learning.