Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs

Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, *Member, IEEE*, David Kaeli, *Member, IEEE*, and Paolo Rech, *Member, IEEE*

***Abstract*—Graphics processing units (GPUs) are playing a critical role in convolutional neural networks (CNNs) for image detection. As GPU-enabled CNNs move into safety-critical environments, reliability is becoming a growing concern. In this paper, we evaluate and propose strategies to improve the reliability of object detection algorithms, as run on three NVIDIA GPU architectures. We consider three algorithms: 1) you only look once; 2) a faster region-based CNN (Faster R-CNN); and 3) a residual network, exposing live hardware to neutron beams. We complement our beam experiments with fault injection to better characterize fault propagation in CNNs. We show that a single fault occurring in a GPU tends to propagate to multiple active threads, significantly reducing the reliability of a CNN. Moreover, relying on error correcting codes dramatically reduces the number of silent data corruptions (SDCs), but does not reduce the number of critical errors (i.e., errors that could potentially impact safety-critical applications). Based on observations on how faults propagate on GPU architectures, we propose effective strategies to improve CNN reliability. We also consider the benefits of using an algorithm-based fault-tolerance technique for matrix multiplication, which can correct more than 87% of the critical SDCs in a CNN, while redesigning maxpool layers of the CNN to detect up to 98% of critical SDCs.**

***Index Terms*—Algorithm-based fault tolerance (ABFT), convolutional neural networks (CNNs), embedded systems, fault tolerance, reliability, safety-critical, soft errors.**

I. Introduction

GRAPHICS processing units (GPUs) have evolved from devices specially designed for graphics rendering, to general purpose accelerators for high-performance computing (HPC) applications and, more recently, to expedite the training and execution of deep learning frameworks. Artificial neural networks (ANNs) are becoming a widely adopted computational approach in many fields, from data mining to pattern recognition, and from robotics to data analytics [1].

Manuscript received January 22, 2018; revised May 18, 2018 and August 14, 2018; accepted October 6, 2018. Date of publication November 15, 2018; date of current version May 28, 2019. Associate Editor: S. Ghosh. *(Corresponding author: Fernando Fernandes dos Santos.)*

F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, and P. Rech are with the Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre 90040-060, Brazil (e-mail: ffsantos@inf.ufrgs.br; pfpimenta@inf.ufrgs.br; cblunardi@inf.ufrgs.br; lucas.kleindraghetti@inf.ufrgs.br; carro@inf.ufrgs.br; prech@inf.ufrgs.br).

D. Kaeli is with the Dana Research Center, Department of Electrical and Computer Engineering, Northeastern University Boston, MA 02115 USA (e-mail: kaeli@ece.neu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2018.2878387

Recently, ANNs have become a very attractive path to deliver highly accurate image classification. *Convolutional* neural networks (CNNs) have, in fact, been deployed in self-driving cars to dynamically identify/classify objects [2], [3]. Major hardware and software vendors have focused their research teams on delivering the fastest hardware/software CNN solution, providing a significant edge over competitors. CNN algorithms exploit the ability of GPUs to support data and thread-level parallelism. GPUs can detect obstacles in a scene in near real time, a critical task in next-generation autonomous vehicles, and are already present in many self-driving cars [4].

Researchers have been overly focused on the performance, while neglecting other critical aspects, particularly reliability. While performance is the key in these applications, reliability needs to be paramount. We cannot tradeoff performance for reliability in safety-critical applications.

In this paper, we analyze the reliability of GPUs executing CNNs, identifying potential hardware/software weaknesses. Next, we apply this knowledge of how faults propagate to program output to propose efficient solutions to improve the reliability of CNNs as run on GPUs. In order to leverage the benefits of a GPU in a vehicle guidance system, the execution must be compliant with the strict ISO26262 requirements [5]. Object detection, being a safety-critical feature, must respect the Automotive Safety Integrity Level D (ASIL-D) standard. ASIL-D requires any component in the system to be able to detect 99% of faults and to have a failure rate limited to 10 failures in time (FIT, errors in 10^9 h of operation). There are several sources of transient errors that can undermine GPU reliability in these systems, including environmental perturbations, software errors, and process/temperature/voltage variations [6], [7]. Radiation-induced soft errors are particularly critical, as they dominate error rates in commercial devices [8].

We investigate how transient errors, on three different NVIDIA architectures (Kepler, Maxwell, and Pascal), impact the reliability of you only look once (YOLO) algorithm and faster region-based CNN (Faster R-CNN) algorithm, which are two of the most popular detection frameworks, and residual network (ResNet), which is the most accurate framework for object classification currently available. While previous work has focused on understanding and improving the reliability of neural networks [9], [10], to the best of our knowledge, this is the first study that targets CNN reliability on GPUs.

We report findings from both beam testing and fault-injection campaigns that assess GPU reliability and identify the causes of

critical errors (i.e., errors that impact detection) and *tolerable* errors. The beam data cover accelerated testing that represents a total of more than 110 000 years of GPU operation. All collected errors are available in a public repository, allowing third-party analysis, and enabling reproductivity of our results [11]. We find that a single fault can spread through the microarchitecture of GPU, affecting several parallel threads, and eventually undermine the neural network's inherent fault tolerance. FinFET technology significantly reduces the silent data corruption (SDC) rate, as we found on the NVIDIA Pascal devices. However, we demonstrate that the more complex structures in Faster R-CNN and ResNet make them more likely to experience SDCs and crashes than YOLO. However, adopting the Faster R-CNN object representation, as well as the ResNet deep pipeline significantly reduces the probability of critical SDCs.

We show that, for CNNs, single error correction double error detection (SECDED) error correcting code (ECC) is useful in some respects, but it is not always effective. Several errors are masked by ECC and do not propagate through the GPU. However, ECC is not able to reduce the number of critical errors. To improve the reliability of CNNs, we adapt YOLO to leverage an already developed algorithm-based fault tolerance (ABFT) strategy for matrix multiplication [12] and evaluate its effectiveness in improving the reliability of CNN. The concept of using an ABFT to harden a CNN arises from the observation that 67% of GPU processing in YOLO, 82% of GPU processing in Faster R-CNN, and 80% of GPU processing in ResNet is spent in matrix multiplication related operations. Using ABFT, we can correct more than 87% of critical errors, clearly outperforming ECC.

Finally, we evaluate error propagation in YOLO when running on a Kepler device using the NVIDIA SASSIFI fault injector [13]. Our results identify the CNNs layers that are most likely to generate critical errors. Based on our study of fault propagation, we provide guidance on how to redesign the maxpool layers to detect up to 98% of critical SDCs. The main contributions of this paper include the following:

1) an evaluation of the reliability on three GPU platforms executing YOLO, Faster R-CNN, and ResNet;
2) an evaluation of the reliability tradeoffs when relying on ECC, versus adopting ABFT, in CNNs;
3) an exploration of transient fault propagation through CNN layers in Kepler architectures;
4) the design of efficient strategies to overcome GPU architectural vulnerabilities and reduce fault propagation in CNNs.

The remainder of this paper is organized as follows. Section II reviews the related work and key concepts of neural network reliability. Section III presents background on YOLO, Faster R-CNN, and ResNet. Section IV discusses the SDC-criticality evaluation. Section V presents beam testing results, and evaluates the efficiencies of ABFT and ECC. Section VI analyzes error propagation in YOLO using SASSIFI. Section VII presents conclusions, and paves a path forward for future work.

II. Related Work

In this section, we review previous work on ANNs, reliability evaluation of GPUs, and available hardening solutions.

A. Fault Tolerance in ANNs

Starting with prior work from the late 1990s, ANNs have been found to be highly tolerant to transient faults [9], [10], [14]. Software fault injection was performed to understand the vulnerability of different networks. Unfortunately, GPUs tend to propagate a single fault to multiple output elements, a behavior not studied in the previous research. Additionally, as the structure and topology of neural networks have evolved significantly in recent years, most prior results cannot be easily extended to today's deep learning complex frameworks. Specifically, neural networks for object detection have the peculiarity of using convolution to reduce the information to be processed [15], a class of computations where reliability has not been studied deeply to date. A pioneering study on the reliability of deep neural networks (DNNs), executed on dedicated accelerators, is presented in Li *et al.* [16], showing that each layer and flip-flop has a different sensitivity to injection. We significantly advance the knowledge on the reliability of DNNs by considering realistic error models (provided by beam experiments) and understanding the propagation of errors injected not only in memory elements but also in computing resources. We consider and compare three commercially available frameworks, and three different GPU architectures. Finally, our hardening strategies do not require costly hardware modifications and are experimentally proven to be very effective and efficient.

Given the results of previous studies, we can expect that the fault tolerance of neural networks will be dominated by subsequent filtering operations. This was suggested using the study of the mathematical algorithms present in general neural paradigms, and by evaluating specific implementations [9], [14].

Prior work has also considered how to improve the reliability of neural networks. Most of the available solutions rely on partial duplication or full duplication (or even triplication) of operations [17]. Some solutions require specific hardware modifications [10], [16]. Both of these approaches are less than ideal for real-time object detection, a task commonly performed in automotive applications, due to processing overhead and added costs.

Some studies have used fault injection during network training [18], [19]. Once the neuron values are adjusted to support errors, the network can maintain the correct classification, even if a fault occurs. The main limitation with this approach is that it is hard to inject a representative set of faults. The fault model set, unfortunately, is determined by the network dimension. For deep CNNs, injecting a statistically significant set of fault models is a huge amount of work, infeasible for most frameworks.

B. Reliability of GPUs and Available Hardening Solutions

As GPUs moved into the HPC market, there has been an increased research studying to improve their reliability. Previous studies have proposed an early evaluation of the reliability of GPUs [20]; they have also evaluated the reliability of GPUs running HPC codes through radiation experiments [21] or fault injection [22]–[24]. Preliminary studies have found some inherent reliability weaknesses in the architecture of GPUs [25], [26]. A single particle-induced fault in the hardware scheduler

or shared memory of a GPU is likely to affect the correctness of several parallel threads, leading to the corruption of multiple output values. Additionally, a single corrupted thread could feed thousands of future parallel threads with erroneous data, again leading to multiple output errors [27]. As a result, the vast majority of transient faults affect multiple output elements in GPUs [21]. These reliability weaknesses arise due to the fact that GPUs were originally designed for graphics rendering, a task where high reliability is less of a concern [28].

In this paper, we advance knowledge on GPU reliability by characterizing how microarchitecture vulnerabilities in a GPU can undermine the reliability of a CNN. Most previous works, in fact, have focused on HPC application reliability on a GPU, leaving the reliability of CNNs unstudied.

We use this new knowledge to address hardware weaknesses in a GPU, designing more robust software. High-end GPUs protect their main storage structures with SECDED ECC. The memory hierarchy includes caches, register files (RFs), and global and shared memories. Some major resources are left uncovered, including flip-flops in pipeline queues, logic gates, block/warp schedulers, instruction dispatch units, and the interconnect network. ECC has been shown to reduce the error rate by one order of magnitude for HPC applications executed on a GPU, but increases the number of crashes [26]. There are several ways to mitigate SDCs in a software, such as using code replication [29]. Unfortunately, while duplication in GPUs has been demonstrated to be even more effective than ECC [26], in real-time systems that come with strict deadlines, the overhead associated with duplication is unacceptable. Some promising results have shown that algorithm-based fault tolerance (ABFT) [12] is a very efficient strategy to protect matrix multiplication [30], [31] and fast Fourier transform (FFT) [32] in GPUs. Single, and many multiple, corrupted output elements affecting matrix multiplication can be corrected in linear time using ABFT [30]. In Section V-D4, we propose to reduce CNN SDC rates by adapting ABFT for general matrix multiplication (GEMM) operations performed during convolution. Finally, in Section VI-C, we take advantage of our knowledge on how errors propagate in CNNs to design a reliable maxpool layer.

III. Background

Next, we discuss background on radiation effects, highlighting intrinsic weaknesses of GPUs. We then review YOLO, Faster R-CNN, and ResNet frameworks.

A. Radiation Effects on Electronic Devices

The primary and secondary impacts of galactic cosmic rays interacting with the terrestrial atmosphere generate a flux of about 13 neutrons/$[(\text{cm}^2) \times h]$ that can reach ground. The flux exponentially increases with altitude [33]. A terrestrial neutron strike may perturb the state of a transistor, generating bit-flips in memory or current spikes in logic circuits that, if latched, lead to an error [34], [35]. A radiation-induced transient error leads to the following:

1) no effect on the program output (i.e., the fault is masked, or the corrupted data is not used);
2) an SDC (i.e., an incorrect program output); or
3) a program crash or device reboot.

Given the shrinking dimensions of CMOS transistors, the pursuit of lower power consumption, and the integration of several resources in a single chip, the probability of neutron-induced faults, in both memory and logic resources, has increased significantly [8]. However, based on physical simulations and experiments using test-chips, it has been reported that FinFET transistors have a much lower neutron-induced error rate as compared to CMOS [36]. On the average, as we measure on live hardware, error rates on CMOS transistors are approximately an order of magnitude higher compared to FinFET transistors.

B. DNNs for Object Detection and Classification

DNNs are one of the most efficient ways to perform image classification [37], segmentation [38], and object detection [2]. Prior work has tried to adopt DNNs for real-time object detection, showing promising results [39], [40]. One of the key steps when using DNNs for object detection is convolution. A kernel filter is convolved with a matrix to extract specific features of the image. The kernel filter slides over the input matrix, multiplying and accumulating products at every position of the input with every position of the kernel. Each block is reorganized as a row of matrix A and the filter kernel is replicated as columns of matrix B. Convolution is then computed as AxB. Other methods consider convolutional algorithms or FFTs. However, they are less efficient than GEMM-based convolutions [41]. As GPUs are highly efficient for accelerating matrix multiplication, they are a great target for CNNs. Specific processors can be designed for DNN, achieving good performance and low power consumption [42], [43]. However, the design of dedicated processors is extremely costly and less than ideal for embedded automotive applications.

Basically, a CNN has several layers that perform convolution on a raw image or feature map (i.e., the output of an upstream convolutional layer). Rectified Linear Units (ReLU), Maxpool, and fully connected layers are the main structures present in most modern CNNs. ReLU layers apply an activation function (removing all negative values) on the feature map to eliminate the input linearity. Maxpool splits the matrix into n block-based regions and propagates only the largest value in each block, reducing the dimension of the processed data by n times. The fully connected layers are conventional ANNs in charge of classifying objects based on the extracted features. As shown in Section VI, maxpool layers mask a significant portion of SDCs, while fully connected layers tend to spread them.

In this paper, we consider three modern frameworks: 1) YOLO [2]; 2) a Faster R-CNN [44]; and 3) a ResNet [45]. *YOLO* is based on *Darknet*, which is an open source CNN written in C and CUDA [2]. Darknet is the first high-accuracy real-time CNN-based system [2]. YOLO performs detection by dividing the input image into $S \times S$ grids. For each grid, YOLO calculates the class probability, the *bounding boxes* (BBs, i.e., potential objects), coordinates and confidence. Performing processing of each frame "only once" delivers detection and classification in real time. Similar to most other CNNs, Darknet

performs four operations to detect objects: 1) layer convolution; 2) a leaky function, similar to ReLU; 3) maxpooling layers; and 4) classification through two fully connected layers. Darknet is composed of 24 convolutional layers interleaved with maxpooling layers.

Faster R-CNN is written in C++ and Python, based on Caffe's [3] deep learning framework. Faster R-CNN execution flow is split into two stages. In the first stage, data pass through 16 convolutional layers. A region proposal network divides the convolutional layer output into regions, and for each region, identifies a set of scores (i.e., object probabilities of that region) and a set of coordinates of BBs. The potential objects (*anchor boxes*) go through a region of interest (RoI) pooling layer, which is a maxpooling operation performed only inside the regions of interest. RoI output is evaluated by two fully connected layers.

ResNet is a CNN for classification written in Lua, based on the Torch7 deep learning framework [46]. ResNet was introduced to increase object-classification precision, without significantly impacting execution time. YOLO and Faster R-CNN would suffer significant classification precision degradation if more than 32 layers were pipelined, due to overfitting. ResNet reaches 95% classification precision by introducing *residual units* between layers. A residual unit combines the feature maps produced in subsequent layers through a residual function. This process is performed repeatedly for all layers in the network, allowing us to use deeper network structures (e.g., 50 to 1000 layers) without impacting precision, and without increasing the algorithmic complexity [45].

ResNet only performs object classification, while YOLO and Faster R-CNN also provide detection (i.e., BBs that highlight the classified objects in the frame). However, the network pipeline and operation performed are similar, once all the CNN share the same structure. There are other differences between YOLO, Faster R-CNN, and ResNet that are likely to impact their reliability. YOLO and Faster R-CNN have four maxpooling layers, while ResNet has one maxpool layer and one average pooling layer (i.e., a pooling operation that propagates the elements average value). Object detection on YOLO and Faster R-CNN is performed using two fully connected layers. On ResNet, only object classification is implemented using a single fully connected layer. YOLO and ResNet have a single execution pipeline (much deeper for ResNet), while Faster R-CNN has only two stages to improve detection precision. As YOLO executes fewer operations, its SDC rate is expected to be lower than that of Faster R-CNN. However, the object representation and the process flow of Faster R-CNN, and the deep pipeline of ResNet, serve to reduce the number of critical SDCs (i.e., SDCs that impact detection), as will be discussed next.

IV. SDC-Criticality Analysis for CNNs

Not all the SDCs are critical in object-detection/-classification frameworks. If the presence of SDCs does not impact detection and classification, SDCs could be considered tolerable. In this section, we describe an error-criticality evaluation for object classification and detection, which we adopt in the next section to evaluate the reliability of CNNs.

The output of YOLO, Faster R-CNN, and ResNet is a vector of *tensors* containing the probability of eligible objects. Objects identified with a sufficiently high probability are classified. In YOLO and Faster R-CNN, a tensor also contains the coordinates of a BB (i.e., potential object), which is then used to describe the detected object. SDCs that modify the probability such that they do not impact the rank of an object, or for a detection framework, change the coordinates of a low-probability BB, are not considered critical.

When radiation significantly impacts classification or detection, we measure the *Precision* and *Recall* of the corrupted output. Recall is the fraction of existing objects that were detected (or classified), even in the event of a radiation-induced error. Precision measures the fraction of the detections produced by the classifier that actually relate to an existing object.

To distinguish between critical and tolerable SDCs for classification, we consider just the probability vector. When an SDC modifies the rank of an object in a frame, we count the number of objects in the fault-free execution that are not correctly classified (i.e., False Negatives, reducing Recall), and how many objects that should not be classified appear in the corrupted eligible objects list (i.e., False Positives, reducing Precision).

A correct classification is not sufficient to guarantee correct detection. To evaluate error criticality for detection, we consider the area of BBs, adopting the methodology used in the image processing community [47]. We consider an object i in a radiation-corrupted output as correctly detected if, for any object j of the radiation-free output, the following condition can be verified:

$$\text{Jaccard\_similarity}(i, j) > T_J \tag{1}$$

where T_J is the acceptance threshold. Otherwise, we consider i as a False Positive (reducing *Precision*). If, for a given object j of the radiation-free output, there is no BB i in the corrupted output, which satisfies this condition, a False Negative is detected (reducing *Recall*).

In the image processing community, T_J is an arbitrary threshold, with $0 \leq T_J \leq 1$. Values of T_J close to 1 force the classifier to be very precise In a reliability context, $T_J = 1$ imposes any radiation-induced modification to the coordinates of BB to be marked as critical. We adopt $T_J = 0.5$ to compare the corrupted output of a CNN with the radiation-free output, following the T_J tradeoff discussion presented in Fawcett *et al.* [47], which is valid, independent of the source of detection imprecision (intrinsic algorithm detection imprecision or radiation-induced corruption). In our experiments, the number of critical SDCs changes by at most 10% when data are parsed using T_J values between 0.3 and 0.7. This means that critical SDCs significantly modify the position of the BB (these data can be retrieved by parsing the results in our online repository [11]).

YOLO describes an object by using the coordinates of its center, height, and width. Faster R-CNN provides the coordinates of two opposite vertices. An error in the object description of YOLO is more likely to modify the Precision and Recall. On the one hand, if the center is corrupted, for instance, we could expect the object position to shift, reducing both Precision (i.e., there is no object in the new position) and Recall (i.e., we missed

the original object). On the other hand, an error in one of the four coordinates of Faster R-CNN will reduce the object area and is not critical as long as (1) is verified. Data presented in Section V-D3 confirm these observations.

The two stages of Faster R-CNN mask errors that affect a region not considered by RoI pooling. Additionally, Faster R-CNN composes objects based on various anchor boxes and anchor scores. Even an error in an RoI that modifies the anchor boxes, and/or the way an anchor box contributes to the object description, could result in a tolerable error. The object, in fact, could be detected with a different shape but, if (1) holds, it could still be noncritical. On the contrary, the detection process of YOLO (performed by the fully connected layers) is directly connected to the last convolution layer. SDC criticality depends on how it modifies the output of the last convolution layer and not on the region of the frame it corrupts (as in Fast R-CNN). To support this statement, we present experimental data in Section V-D2.

For ResNet, we only consider the Precision and Recall of classification. We anticipate that, for YOLO and Faster R-CNN, only 1% of the critical SDCs for classification are not critical for detection. This is to be expected, as convolution data are used to perform both classification and detection. An error in the network is then likely to affect both detection and classification.

We use the PASCAL VOC 2012 [48] and the Caltech Pedestrian datasets [49] as input for our study. The PASCAL VOC dataset has been used extensively to evaluate detection capabilities of YOLO [2]. The Caltech dataset is used frequently in today's automotive research. We mark a critical error as any error affecting the Precision and/or the Recall of a single frame. While a self-driving system could be tuned to base decisions on several frames, there are situations in which such an implementation may be impossible. Current car cameras support 30 frames/s, which means that, in the best case, a car driving at 75 mi/h will travel for about 11 ft before receiving 3 frames to analyze.

V. Neutron Beam Experiments

In this Section, we present the three GPUs evaluated, and our experimental data. In Section VI, we analyze architectural and application reliabilities with fault injection.

A. Devices Under Test

We consider NVIDIA GPUs from three different product families: 1) Tesla K40; 2) Tegra X1; and 3) Titan X. While Tesla K40 and Titan X are not designed for automotive applications, they are selected to evaluate the resilience of the Kepler and Pascal microarchitectures to soft errors while running CNNs.

Tesla K40 supports the Kepler instruction set architecture (ISA), and is fabricated in a 28-nm standard CMOS technology. This model has 2688 CUDA cores divided across 14 streaming multiprocessors (SMs). The RFs, shared-memory, and caches are SECDED ECC protected, while the read-only data cache is parity protected.

Tegra X1 is an embedded version of the Maxwell GPU, and includes an ARM quad-core CPU. It is fabricated in 20-nm standard CMOS technology. Tegra X1 GPU has 256 CUDA cores divided in 2 SMs.

TABLE I
Frames per Second

	X1	Tesla K40	Titan X
		Object detection	
YOLO [fps]	3.07	10.43	15.46
Faster R-CNN [fps]	0.41	3.65	7.83
		Classification	
Resnet	–	34.68	–

Titan X is designed with the latest Pascal architecture, in 16-nm FinFET technology. Titan X has 3072 CUDA cores split across 24 SMs. The Pascal architecture includes support for *mixed precision*. Operations that do not need high precision (e.g., processing camera data or other sensor data) are performed in FP16 (half precision). While FP16 mode does not reduce the detection quality of CNN; it can impact reliability.

The different transistor layouts will impact the probability of a particle generating a bit-flip(s) or logic error(s) [36], while microarchitectural differences will affect the way low-level faults propagating to a visible program output. The experimental part of this study evaluates both aspects.

Table I lists the *frames per second* of each device. Faster R-CNN experiences longer delays due to its inherent complexity as compared to YOLO. This is particularly significant on the Tegra X1, where executing Faster R-CNN is compromised by the limited computational resources (i.e., two SMs). ResNet is much faster than both YOLO and Faster R-CNN, even if ResNet is almost $7x$ deeper, as it only performs object classification. Both object detection and classification use matrix multiplication based convolution. However, for implementing detection, convolution is much more complex (i.e., it uses larger kernels and feature maps).

B. Experimental Setup

Our radiation experiments were performed at the ChipIR Facility, Rutherford Appleton Laboratory, Didcot, U.K., and at the Los Alamos Neutron Science Center (LANSCE) Facility, Los Alamos National Laboratory, Santa Fe, NM, USA. We test each device and configuration at both ChipIR and LANSCE. FIT rates were similar (within a margin of error) across facilities.

Fig. 1 shows part of our setup at ChipIR. A total of three Tesla K40s, one Titan X, and six Tegra X1s were irradiated. The onboard dynamic random acess memory (DRAM) was not irradiated. DRAM sensitivity has already been extensively studied [50], [51], and is beyond the scope of this paper.

ChipIR and LANSCE provide a neutron beam suitable to mimic the atmospheric neutron effects in electronic devices. The available neutron flux was about six to eight orders of magnitude higher than the terrestrial flux (13 $\mathrm{n}/(\mathrm{cm}^2 \times \mathrm{h})$ at sea level [33]). Each device was tested for at least 100 h, which is equivalent to more than 100×10^6 h of normal operation (i.e., 11 500 years). As we tested 10 devices, we are reporting data that cover more than 110 000 years of GPU operation in a natural application. Since the terrestrial neutron flux is low, in a

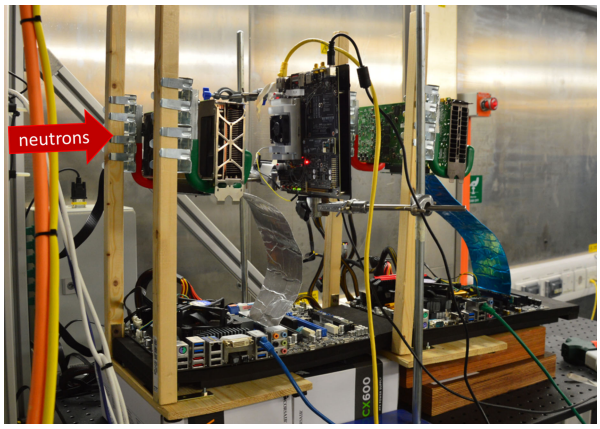


Fig. 1. Radiation test setup at ChipIR.

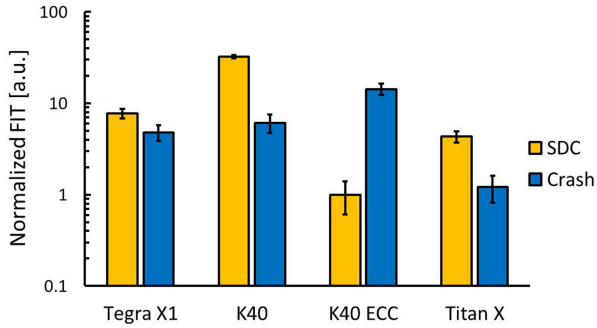


Fig. 2. Normalized FIT for GEMM.

realistic application, it is highly unlikely to see more than a single corruption during program execution. We have carefully designed the experiments to maintain this property (observed error rates were lower than 10^{-3} errors/execution). Experimental data, then, can be scaled to the natural environment without introducing artifacts.

The host processor initializes the test, sending preselected inputs to the GPU, and gathers the results of kernel execution. Each result is compared with a precomputed golden output. When a mismatch is detected, the execution is marked as affected by an *SDC* and the data are stored for postprocessing. Both software and a hardware watchdog are used to detect *crashes*.

Experiments aim at measuring the *cross section*, obtained by dividing the number of observed SDCs or crashes by the received neutron fluence (neutrons/cm^2). Multiplying the cross section with the expected neutron flux at which the device will operate produces the FIT rate, which is the sole reliability metric used in ISO26262 [5]. As FIT is a business-sensitive metric, we report only relative values shown in *arbitrary units*. We can say, however, that all reported SDC rates for CNNs are higher than the 10-FIT limit imposed by ASIL-D. The hardening solutions we propose become essential to reduce CNN FIT rates, making them almost compliant with ASIL-D. FIT does not depend on the execution time (nor frames/s) [8]. We can consider performance-reliability tradeoffs by dividing the frames/s by the FIT rate, gathering the number of correctly computed frames between failures [52].

C. GEMM Reliability

Beam testing results on GEMM kernel serves as a baseline to characterize the reliability on the three different GPU microarchitectures and provides important insights into the reliability of CNNs. It is worth noting that convolution could also be FFT based [41]. A reliability analysis for FFTs has already been presented in Pilla *et al.* [53].

Fig. 2 shows the SDC and crash normalized FIT rates for GEMM (without mixed precision), presented with 95% confidence intervals. Values are relative to the SDC rate of the Tesla K40 with ECC enabled. We selected matrix sizes that saturated the resources of each device ($2^{10} \times 2^{10}$ for Tegra X1, and $2^{11} \times 2^{11}$ for Tesla K40 and Titan X). As device resources are saturated, data in Fig. 2 can be used to compare the reliability of GEMM across architectures. Smaller matrices should lower the FIT rate given the number of unused resources. Multiplying $2^{11} \times 2^{11}$ matrices on Tegra X1 would take too long to guarantee that there would be only a single corruption per execution.

From Fig. 2, it is clear that GEMM reliability depends on the device. The SDC rate for Titan X is only 14% of that for Tesla K40, and about half of the SDC rate for the Tegra X1. This result proves that, on live hardware, previous studies have reported a much improved reliability for FinFET technology as compared to planar CMOS [36]. The probability of resource corruption on Titan X is then expected to be much lower than on Tesla K40 and Tegra X1. Still, in accordance with previous studies [26], crash rates are lower than SDC rates for all configurations, except for Tesla K40 with ECC enabled. ECC increases the crash rate by around 50% for Tesla K40. The SECDED ECC of GPU triggers an application crash when a double bit error (i.e., a detected unrecoverable error) is detected. This same double bit error may (or may not) lead to an SDC when ECC is not used. It is worth noting that with the shrinkage of transistor dimension, the percentage of neutron-induced faults that generate multiple cell upsets will increase—for the current technology node, the increase is 20%–30% [54]. Some of these multiple faults lead to double bit errors, inducing a crash when ECC is enabled.

An additional aspect of the reliability of a GPU, which can be gleaned from inspecting the output values of neutron beam experiments, and would not be possible using GPU-Qin or SASSIFI, is the degree of mismatch in the output values. Small variations in an output value could have little or no impact at all in terms of CNN detection or classification (e.g., small changes in the shape of the object would not be considered critical, or small probability variations). Fig. 3 illustrates how varying the acceptable error margins affects the SDC FIT rate of GEMM. We plot how much the GEMM SDC FIT rate changes (vertical axis) when we increase the acceptable error margin from 0% to 10% (horizontal axis). The different FIT rate reductions are symptomatic of how physical-level faults propagate through the GPU microarchitecture. Titan X and Tesla K40 show similar trends, suggesting that the lower FIT rate for Titan X should be attributed mostly to the reliability benefits of FinFET.

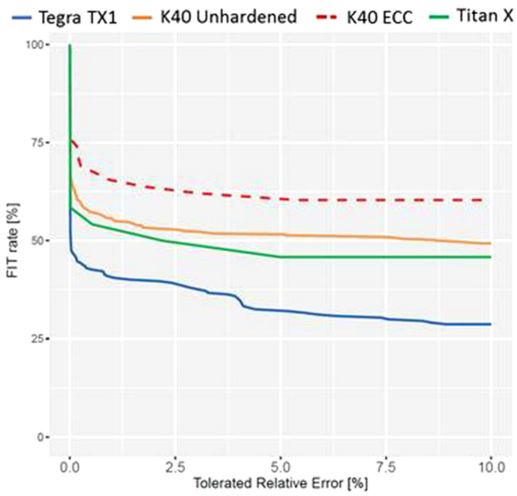


Fig. 3. FIT reduction as a function of relative error tolerance.

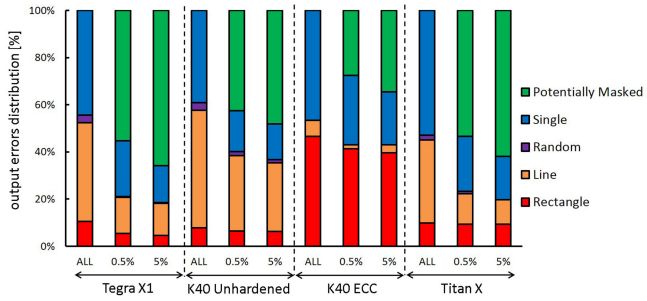


Fig. 4. SDCs spatial distribution in GEMM.

About 50% of the corrupted execution on Tegra X1, and 40% on Tesla K40 and Titan X, could be treated as tolerable if we use a 0.01% error margin. The ECC-hardened Tesla K40 only sees a 25% FIT reduction when using this less stringent threshold. While ECC reduces the number of SDCs, faults in the unprotected resources of Tesla K40 produce errors that are more severe than the faults masked by ECC. In fact, SECDED ECC corrects only single-bit faults that affect main memory structures. These faults are likely to lead to less critical SDCs, as compared to faults in core logic or in pipeline registers (which are not protected by ECC), as the latter are likely to significantly impact the program output, corrupting more than a single bit. As a consequence, ECC only supports the simplest of data fault models, increasing the portion (but not the absolute number) of critical SDCs. As discussed in Section V-D4, this is a reason why ECC should not be enabled for CNNs.

We further inspect GEMM reliability, showing the spatial distribution of output errors, as proposed in Oliveira *et al.* [21], which used a distribution that classifies errors based on their coordinates. That is, Fig. 4 shows the percentage of corrupted executions that are affected by single, line (i.e., multiple corrupted elements on the same row/column of the output matrix), rectangle (four or more elements distributed in a rectangle/square), and randomly distributed errors. We also show the spatial error distributions for 0.5% and 5% acceptable error margins. When a faulty execution has all of its corrupted elements inside the tolerance margin, we tag it as *potentially masked*. In the vast majority of cases, GPU corruption affects more than a single output element. It is worth noting that multiple corrupted elements are not caused by multiple impinging particles, but instead inherent in the GPU computation to *spread* low-level transient faults across multiple output elements. If we consider multiple corrupted elements, the reliability of the neural network is going to be impacted.

TABLE II
NORMALIZED FIT

YOLO	SDC	Critical SDC	Critical SDC %	Crash
Tegra X1	12.02	7.21	59.98%	36.06
Tesla K40	22.91	1.83	7.99%	29.59
Tesla K40 ABFT	22.43	1.17	5.21%	29.99
Tesla K40 ECC	4.83	2.94	60.87%	33.79
Titan X	1.75	0.31	17.71%	3.73
Titan X ABFT	1.00	0.125	12.5%	2.17
Faster R-CNN				
Tegra X1	646.21	646.21	139.21%	645.22
Tesla K40	116.50	6.38	5.48%	118.12
Tesla K40 ECC	14.74	3.69	25.03%	166.38
Titan X	210.99	0	0%	113.03
ResNet				
Tesla K40	82.15	6.54	7.96%	88.64
Tesla K40 ECC	23.18	3.60	15.5%	129.93

Interestingly, most of the potentially masked executions are those affected by a single corrupted element, while the number of rectangle errors remains almost unchanged. We can conclude that those errors that spread through the GPU microarchitecture and the GEMM program are the same ones that have a greater difference from the expected value. Unfortunately, as shown in Section VI-B, while most single errors do not propagate through the maxpool layers of the CNN, rectangle SDCs can be more severe for CNN reliability. We found that the spatial distribution of errors, which are the errors likely to propagate through a CNN, are also the values with the larger difference from the expected values, and thus, are the ones more likely to impact detection.

Rectangle errors, as we said, are potentially the most severe for CNNs. Rectangle errors in GEMM are caused by faults that impact the scheduling or the execution of multiple threads in an SM. GEMM increased execution efficiency, as compared to a naive matrix multiplication implementation, since it can divide the input matrices into chunks that fit nicely in the cache of a SM, reducing memory latency. If a fault can cause a thread to be incorrectly assigned or scheduled to a SM, or if some threads fail to synchronize, the whole SM output matrix portion is likely to be corrupted, leading to a rectangular error. Errors in memory elements protected by ECC (e.g., registers and caches) have been reported to manifest into either single corrupted element or line errors [30]. When ECC is turned ON, single and line errors (which are less critical for CNNs) are corrected, but all the other errors (including rectangular errors) are not corrected. As a result, the percentage of rectangular errors increases when ECC is enabled.

D. CNN Reliability Evaluation

Table II lists, and Fig. 5 shows (in log scale), the normalized FIT rate of YOLO and Faster R-CNN executed on Tegra X1,

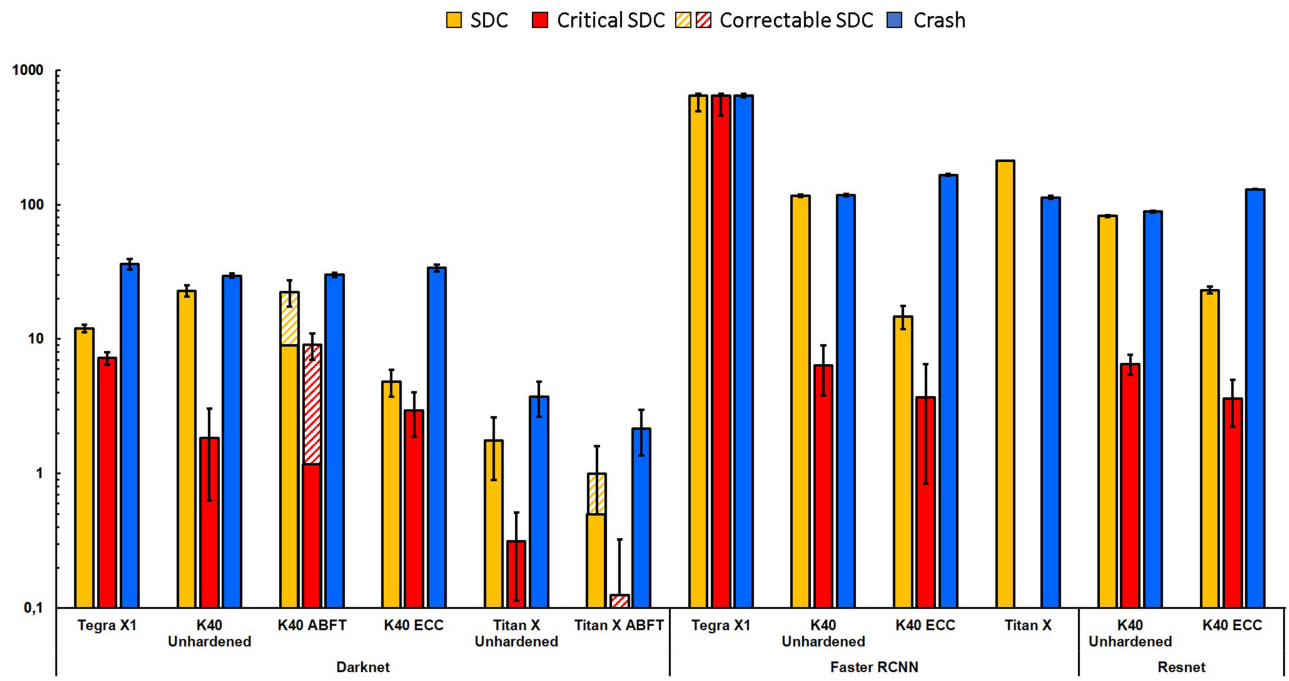


Fig. 5. Normalized FIT for YOLO, Faster R-CNN, and ResNet.

Tesla K40, and Titan X, and ResNet executed on Tesla K40. We could not test ResNet on the other devices due to beam time limitations. All reported values are relative to the SDC FIT rate for the ABFT-protected Titan X. The third column of Table II shows the percentage of SDCs that are critical (i.e., that impact classification/detection). Experimental data are presented with a 95% confidence interval. We report the normalized FIT for SDCs (any error that appears in the output), Critical SDCs (errors that reduce either Precision or Recall and, thus, impact detection), and crashes. We test the Tesla K40 with ECC enabled and disabled (Unhardened), and ABFT-protected YOLO on Tesla K40 and Titan X.

1) Crashes: Crashes are more probable than SDCs for all of the tested configurations, except for Faster R-CNN as run on Titan X. This result is in contrast with the trends observed when running GEMM, as discussed in Section V-C, and for HPC applications, as presented in prior work [26], [55], where the crash rate is lower than the SDC rate. Neural networks have an inherently higher SDC masking capability than most other codes [18], [56], [57]. Crashes, on the contrary, have a component that depends on the GPU control logic sensitivity, and on the CPU–GPU context change, which does not benefit from the SDC tolerance of neural networks. On object-detection operations, all GPU kernels have a high level of reuse that requires several device–host synchronizations. A transient fault during those synchronizations could potentially result in a GPU crash. For the same reason, Faster R-CNN, which requires a much larger number of CPU-GPU synchronizations, shows a $10\times$ higher crash rate than that shown by YOLO.

ECC has the drawback of increasing the crash rate for Faster R-CNN, ResNet, and YOLO. However, the ECC Crash rate increases by approximately 20%, much lower than the 50% observed for GEMM in Section V-C. As discussed in Section VI-B, a significant portion of crashes is caused by corruptions of registers holding index values, where ECC can mask these errors.

Finally, the higher crash rate for Tegra X1 is not surprising as the embedded host processor (the ARM) was also irradiated, whereas the hosts for Titan X and Tesla K40 were carefully placed away from the beam.

Our experimental data attest that for YOLO, Faster R-CNN, and ResNet, radiation-induced crashes are a serious limitation of GPU reliability. A detection system that uses GPUs in vehicles must have a watchdog or other fast crash detection system to ensure availability and avoid missed deadlines. Even if crashes are more frequent than SDCs, they could be considered less critical since crashes could be, at least, detected [58]. However, the system must be able to recover before causing any harm to the environment [59].

2) SDCs: SDC trends across GPUs are shown in Fig. 5. The trend for YOLO is similar to the trend found in Fig. 2 for GEMM. YOLO does not take advantage of the mixed precision available on Titan X. For YOLO, we find similar insights and come to similar conclusions as we did for GEMM in Section V-C: SDC Rates are Related to Physical Transistor Implementations and Microarchitectural Differences.

As expected, from the analysis presented in Section III-B, the complex structure of Faster R-CNN and the deeper network of ResNet increase their SDC rate as compared to that of YOLO (more than $10x$ higher for ResNet and Faster R-CNN on Tesla K40 and more than $100x$ for the Faster R-CNN on Tegra X1 and Titan X).

Faster R-CNN employs mixed precision libraries for Titan X (unavailable for Tesla K40 and Tegra X1), which could exacerbate the SDC rate. Mixed precision libraries increase the number of operations executed concurrently. If a 16-b number or operation is corrupted, the error is more likely to propagate

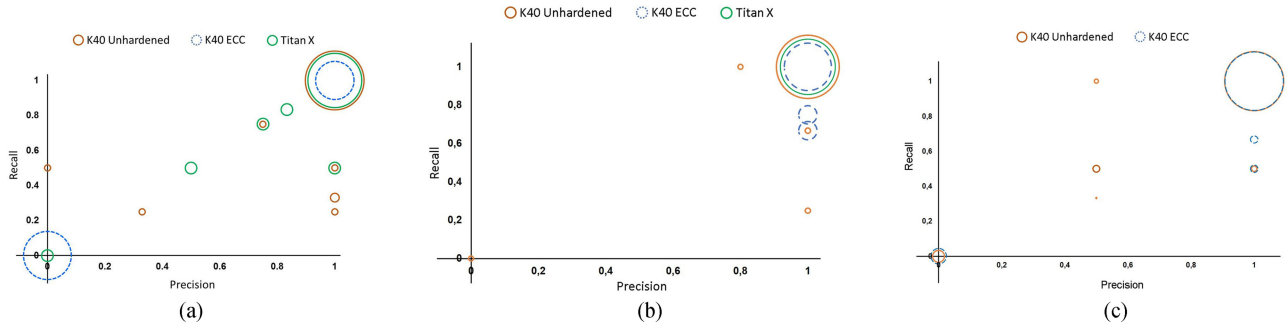


Fig. 6. Precision and Recall for: (a) YOLO, (b) Faster R-CNN, and (c) ResNet. A Precision lower than 1 could potentially lead to unnecessary car stops (i.e., inexistent objects detected) while a Recall lower than 1 could lead to accidents (i.e., undetected objects). The diameter of the bubble is proportional to the percentage of SDCs, with the Precision and Recall values indicated by the center coordinates of bubble. Tegra X1 results are not shown.

than for an error in a 32-b number. Errors in the least significant digits of a 32-b number are much less severe than errors in the least significant digits of a 16-b number. Additionally, there is a higher chance to corrupt the exponent in an FP16 representation versus that in an FP32 representation ($5/16$ versus $8/32$). While beneficial to the performance, mixed precision may increase the device FIT rate. As discussed in Section V-B, the FIT metric does not consider execution time. To evaluate the tradeoff between performance benefits and increased error rates, we can divide the frames/s (see Table I) by the FIT rate (see Table II), providing the number of frames processed by each architecture between failures. While the YOLO Titan X can correctly compute $20x$ more frames than Tesla K40 Unhardened, Titan X correctly computes just 20% more frames than Tesla K40 for Faster R-CNN. Recent research has been focused on the option of using fixed-point or integer operations for object detection [60]–[62]. Unfortunately, it has been shown that faults in integer data are more likely to impact the execution of a GPU than faults in FP32 data [63]. Reducing the operation precision will improve the performance, but it increases the chance of GPU errors, making hardening solutions, such as the class of problems presented in Sections V-D4 and VI-C, even more important for future frameworks.

Fig. 5 shows that ResNet has a higher FIT rate than that of YOLO; although the rate is similar when compared to Faster R-CNN. The higher efficiency and simpler convolution are insufficient to compensate for the higher error rate associated with a deeper neural network. Our results suggest that the higher the precision, unfortunately, the more likely that output errors will be seen. Hardening solutions, as presented in Sections V-D4 and VI-C, are then mandatory for safety-critical applications.

Tegra X1 has the highest SDC rate for Faster R-CNN, mainly due to the complexity of the framework. On Tesla K40, the execution time for Faster R-CNN is only $2.7x$ longer than that for YOLO, while on Tegra X1 it is more than $7x$ longer than that of YOLO (see Table I). For Faster R-CNN, due to the limited performance of Tegra X1, data remain in caches and registers longer. These data are exposed and are critical since they are likely to be used in future operations. The longer these data are exposed, the higher the SDC rate.

3) Critical SDCs: For all the tested CNNs, a significant number of the radiation-induced errors that propagate to the output are not considered critical, as they do not impact the detection (see Fig. 5). As discussed in Section IV, the main reasons why critical SDCs occur less often versus SDCs are as follows.

1) Some SDCs modify the object probabilities in such a way that the ranking is unaffected.

2) Not all output data will be used for detection.

3) Even if an identified object is corrupted, its shape could be sufficiently similar to the expected one, and thus, be correctly detected.

Additionally, all CNN operations use floating point operands. Some of those floats represent the object coordinates, which need to be cast into integers. Errors affecting the low-order bit positions are not expected to impact detection.

For object detection, the percentage of SDCs that are critical is much lower for Faster R-CNN than that for YOLO, independent of the architecture. For YOLO, the percentage of critical SDCs is 8% for an unhardened K40, 61% for the K40 with ECC ON, and 18% on the Titan X. For Faster R-CNN, the critical SDCs are 5% for an unhardened K40 and 25% for the K40 with ECC ON. On Titan X, for Faster R-CNN, none of the hundreds of observed SDCs had an impact on detection. This results from the detection mechanism and object representation used in Faster R-CNN. Several anchor boxes are used to define an object, and a corruption in one of the coordinates of a vertex does not significantly impact the detected object shape.

To better understand the different behaviors of YOLO and Faster R-CNN Fig. 6(a), (b), and (c) shows the Precision and Recall for all observed SDCs in Tesla K40 and Titan X [Tegra X1 is not shown since most of critical SDCs would be located at (0, 0)]. Precision is shown on the x-axis and Recall on the y-axis. Precision <1 means that additional objects are detected (just classified, for ResNet). The lower the Precision value, the higher is the probability of having unnecessary car stops. A Recall <1 means that some objects are missed, which could potentially lead to accidents. To provide information about the distribution of observed faults, in Fig. 6 we use bubbles, whose diameter is directly proportional to the percentage of data samples with the value of Precision and Recall indicated by the center of the bubble. The larger bubbles in the top right side of Fig. 6 indicate that many errors have a minor impact on detection, while bubbles close to the origin indicate that the observed errors significantly impact detection.

Tesla K40 and Titan X follow a similar trend. For YOLO, both Precision and Recall are gradually reduced, until reaching 0 [data points are located in the diagonal of Fig. 6(a)]. For some corrupted executions, Precision is maintained at 1, while Recall is reduced. This is the effect of YOLO object description, which uses the coordinates of its center that, if corrupted, shifts its position reducing Precision and Recall at the same rate. Errors that reduce only Recall are caused by detection probability modifications (objects are not detected as their probability has been lowered).

For Faster R-CNN, most of the critical errors on Tesla K40 and Titan X preserve a Precision of 1, reducing just Recall [there are no data points in the diagonal in Fig. 6(b)]. Corruption in the coordinates of one of the two opposite vertices in Faster R-CNN object description just changes the area of the object, eventually reducing Recall but not Precision. Representing the object with vertex coordinates seems to be a more reliable choice.

For Tegra X1 (not shown), unlike Tesla K40 and Titan X, most of its critical errors have a Precision and Recall value equal to 0. This is probably due to the more highly integrated embedded architecture of Tegra X1, which requires the same hardware to perform several operations for the same layer. A critical error is then likely to significantly impact the detection.

As shown in Fig. 6(c), ResNet follows the same trend as seen in YOLO and Faster R-CNN. This result confirms, once again, that object classification and object detection exhibit very similar behaviors when corrupted.

4) ECC- and ABFT-Efficiency Evaluation: Fig. 5 and Table II show the error rates for Tesla K40 with ECC enabled. Both Tesla K40 and Titan X are protected with the ABFT strategy. YOLO executed on Tesla K40 with ECC enabled is less reliable than in the unhardened Titan X, which is a symptom of the poor resiliency provided by ECC. Based on the GPU-Qin analysis in [27] we know that ECC does not mask all the faults as error in computing elements could propagate to the output. Our beam tests provide, additionally, the realistic probability of experiencing an SDC when ECC is enabled or not, which is the only way to evaluate the effectiveness of ECC. A novel and worrying insight we derive from our beam tests is that ECC does not reduce (or reduce only slightly) CNN critical SDC rate.

The SDC rate for YOLO, Faster R-CNN, and ResNet on the ECC-protected Tesla K40 is 25%, 14%, and 28% of the SDC rate seen for an unhardened Tesla K40, respectively. ECC is not as effective on these workloads as it is for other codes, mainly because neural networks are intrinsically resilient to data errors. Similar to results from previous studies [26], Fig. 2 shows that ECC reduces the GEMM SDC rate by about one order of magnitude. ECC is less effective in protecting a CNN, as some of the SDCs that are masked by ECC would not have affected the CNN execution. A valuable insight from our results is that ECC does not reduce the number of critical SDCs for YOLO, Faster R-CNN, or ResNet.

The portion of SDCs that impact detection is less than 8% for an unhardened Tesla K40, and can be more than 60% in the ECC-protected Tesla K40 (see the third column of Table II). Additionally, Fig. 6(a) shows that critical SDCs (with ECC enabled) significantly impact Precision and Recall. This confirms that most of the critical SDCs come from faults outside of the main memory structures. Our data strongly suggest that faults in caches or registers are not as severe for CNNs, as compared to corruptions in logic, the scheduler, or flip-flops. Fault injection in Section VI-B confirms our conjecture.

Figs. 3 and 4 help to explain why ECC is not very effective in reducing SDCs and most critical SDCs in CNNs. ECC reduces the absolute number of SDCs, but has the side effect of increasing the portion of rectangle errors, which, as described in Section VI-B, are more likely to propagate through CNN layers and affect detection. Additionally, the average difference between a corrupted element's value and the expected value is higher when ECC is enabled (see Fig. 3). It is then more likely for the faults not masked by ECC to significantly impact the correctness of CNN.

Since ECC does not seem to be effective for CNNs, we leverage ABFT to protect YOLO, protecting matrix multiplication operations, as described in Huang *et al.* [12], and extended in Rech *et al.* [30], to correct line and random errors in $O(N)$ time. For FFT-based convolutions, a promising ABFT able to detect more than 80% of faults has been describe in Pilla *et al.* [53]. Unfortunately, it is not possible to implement and evaluate ABFT on Faster R-CNN and Resnet, as they are built with NVIDIA proprietary cuDNN libraries, based on Caffe and Torch, respectively. Nevertheless, the GEMM core used to perform convolution in Faster R-CNN and Resnet is the same algorithm used in YOLO. The portion of SDCs affecting GEMM that ABFT can correct is going to be very similar between the three frameworks. Actually, since the percentage of GEMM operations is higher in Faster R-CNN and Resnet (82% and 80%, respectively) than that in YOLO (67%), ABFT could even be more effective for the former frameworks.

To adapt ABFT for our CNNs, we decided not to add a dimension to the input matrices for the checksum, but instead to *substitute* the last row of the first input matrix and the last column of the second input matrix with column/row checksums, respectively. The GEMM kernels, in fact, are tuned to fully use caches and registers. Adding an extra dimension would not only compromise execution time, but also significantly increase data movement and memory latency. This would result in a different behavior under radiation not solely related to ABFT. It is worth noting that the evicted row/column does not significantly reduce YOLO detection capabilities (differences from the unhardened and ABFT-protected versions are lower than 10%). We believe that performing the training of YOLO with the evicted rows/columns would reduce the detection differences. To focus on the efficacy of our hardening strategy, we maintain the same weights (i.e., the same nonevicted data) to keep the protected and unprotected versions similar.

The ABFT procedure does not significantly affect SDCs or crash error rate of YOLO. However, ABFT is capable of correcting about 60% of the SDCs on Tesla K40, and 50% on Titan X. If we consider only the critical SDCs, an ABFT-protected YOLO is more resilient than an ECC-protected version. This is because ABFT corrects all the errors that affect GEMM computation (including, but not limited to, memory errors). As shown in Fig. 4, 80% to 90% of the observed errors for GEMM are single, line, and random errors, which ABFT can correct.

As noted earlier, on GPUs, the ABFT code is run in parallel and will be executed in linear time. However, since ABFT performs multiplication/accumulation, it can be implemented using the same hardware used for matrix multiplication. That is, almost no hardware changes are required to implement the proposed hardening. ECC, on the contrary, has a logarithmic memory cost.

VI. Fault-Injection Analysis

A. SASSIFI Fault Injector

In this Section, to support the findings discussed in Section V, and to better understand fault propagation in CNNs, we present the results of a fault-injection study on YOLO executed on K40.

For our analysis, we use SASSIFI, which injects transient errors in the GPU's ISA visible state, including general purpose registers, memory values, predicate registers, and condition registers [13]. Prior work used GPUQin [23] to evaluate CUDA benchmarks reliability. We choose to prefer SASSIFI owing to the following reasons.

1) It is much faster than GPUQin.

2) It injects faults at a much lower level, allowing for a more detailed analysis of the reliability of GPU.

As SASSIFI is based on the SASSI instrumentation tool. It does allow us to inject faults in Faster R-CNN and Resnet, given that they are based on NVIDIA proprietary cuDNN libraries. Additionally, SASSI is not yet available for the newest CUDA versions, which prevents us from performing faults injections on Pascal and Maxwell devices. Nevertheless, Pascal and Maxwell devices share the same ISA, which is very similar to the Kepler ISA. The Maxwell ISA includes 20% more instructions than the Kepler ISA, with most of the added instructions related to mixed precision operations. As YOLO does not use mixed precision, it is reasonable to believe that the results obtained with SASSIFI on the Kepler device are applicable to Pascal and Maxwell architectures. If Faster R-CNN and Resnet are implemented without mixed precision, their fault propagation behavior is going to be similar to that of YOLO, as all frameworks share the same structure: a cascade of convolutional layers (all of which use the GEMM core) interleaved with MaxPool and ReLu layers. We will consider the impact of mixed precision on CNN reliability in future work.

We inject faults both at the instruction output (INST) and in the RF. INST injections measure the program vulnerability factor (PVF), which is the probability that a transient error that modifies the result of an instruction will propagate to a program visible output [64]. RF injections measure RF architectural vulnerability factor (AVF) and are used to study how applications propagate errors in memory elements [65].

We inject a cocktail of fault models composed of single bit-flips, multiple bit-flips, and zero/random values [13]. All types of faults can be injected on a single thread or in the threads in a warp. We inject more than 2000 faults at each error site (INST and RF) while running YOLO, which are sufficient to guarantee worst-case statistical error bars at 95% confidence level to be at most 1.96%. As shown in [13] and [66], injecting a higher number of faults would not significantly change our statistics.

B. Error-Propagation Analysis

As described in Section III-B, YOLO is based on *Darknet* CNN, which is composed of 24 convolutional layers interleaved with 4 maxpooling layers (Layers 1P, 3P, 8P, and 19P), 1 local layer (28L), 1 dropout layer (used only in the training phase and not considered in our study), and 2 fully connected layers (30F and 31F). The input of the first layer is a resized raw image, which is a $448 \times 448 \times 3$ matrix. From the second layer through the local layer, data are managed as three-dimensional (3-D) matrices, with gradually reducing matrix sizes. The output is a $7 \times 7 \times 35$ tensor vector, encoding the potential object predictions and their confidence scores.

To track faults propagation through CNN layers we create a utility that captures all of the outputs from each of the 31 layers (dropout is excluded). This feature adds a small execution overhead, but does not modify the execution flow. When a fault is injected, we compare each layer's data with the fault-free data. We can then identify the layer in which the fault occurred and how the fault modifies the data matrix in each layer (distinguishing between single, line, rectangular, cubic, or random errors). We can also track how that fault propagates through the CNN pipeline until the fault is masked or reaches a visible program output.

Fig. 7 shows the AVF and PVF values for injections in the RF and INST sites, respectively. For each layer, we measure the following probabilities that an injection:

i.) crashes the application; or

ii.) propagates to an output (SDC); or

iii.) modifies detection (critical SDC).

Fig. 7 also shows how injections in a layer manifest at the layer output (single, line, rectangular, cubic, and random).

The AVF and PVF rates for crashes are very similar across layers (there is less than a 5% variation), so we show only the overall crash AVF and PVF rates. It is worth noting that injections cause the vast majority of crashes in RF mode (the AVF is 0.74, the PVF is 0.09), mostly because of injections in index registers and addresses for memory stores. ECC can help in masking those crashes. Unfortunately, ECC is not able to guarantee safe crash protection, as errors in unprotected structures (control logic, GPU–CPU synchronization, etc.) dominate the crash rate.

INST injections are much more likely to generate SDCs or critical SDCs than RF injections. RF injections from Layers 18 to 31 generate zero critical SDCs. On the average, the PVF is 0.38 and the AVF is 0.08. The fact that memory faults (RF) are much less likely to generate SDCs than errors injected in INSTs supports our claim that ECC is not very effective in protecting CNNs.

Injections at the input of the two fully connected layers (30F and 31F) do not produce SDCs, confirming the intrinsic fault tolerance of the ANN. Most of the SDCs at the CNN output come from faults impacting a convolution layer. Our detailed analysis of GEMM reliability across microarchitectures, presented in Section V-C, can then be used to help explain the reliability of CNNs.

The 24 convolutional layers, each running the same kernel, have the same *standalone* AVF and PVF. However, as shown

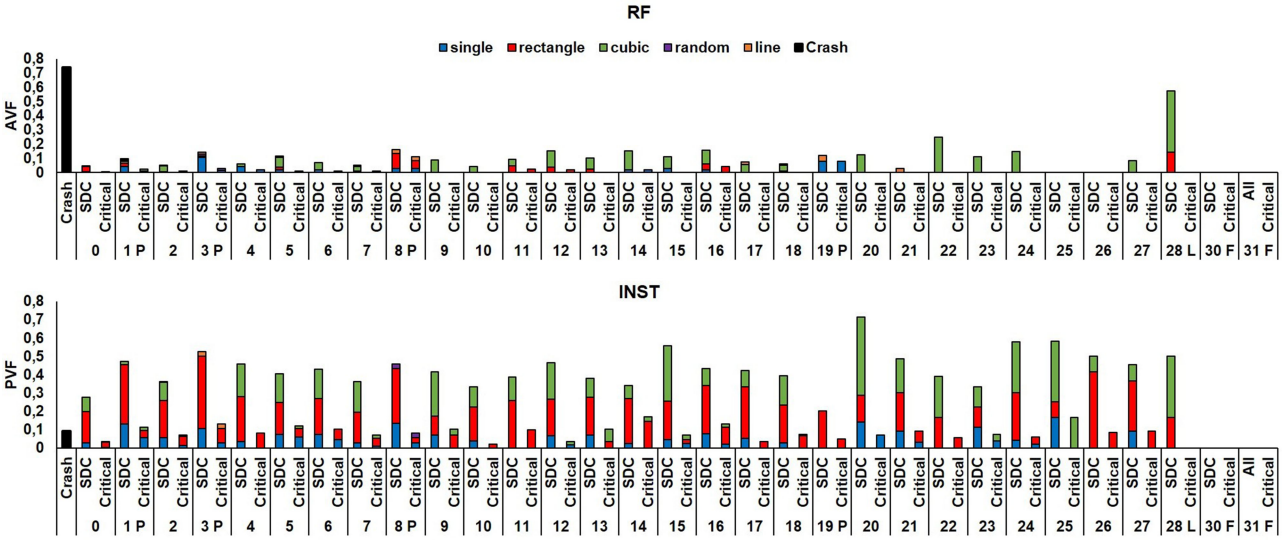


Fig. 7. AVF and PVF for crashes and, of each layer, for SDCs and critical SDCs.

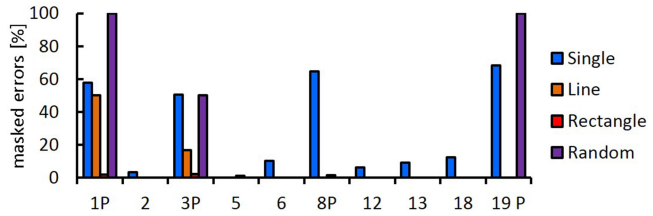


Fig. 8. Breakdown of masked errors.

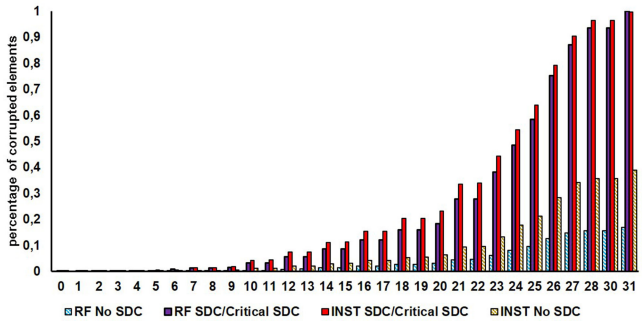


Fig. 9. Average percentage of corrupted elements at the output of each layer.

in Fig. 7, the probability of an injection to affect the network output depends on the position of the layer. Both AVF and PVF decrease before the maxpooling layers (1P, 3P, 8P, and 19P in the Figure) and then increase in downstream layers. Maxpooling discards three-fourth of the data from the previous layer, eventually masking SDCs. As shown in Fig. 8, most of the single errors and random errors do not propagate through the maxpooling layers, while line errors are masked only if they occur in the first layer, and rectangle and cubic errors always propagate. This is in agreement with Fig. 7, that shows a much higher probability of rectangle and cubic errors resulting in SDCs. Comparing Figs. 8 and 4, we can deduce that most of the errors corrected by ECC (single and line) would have been masked by maxpooling layers.

Our fault-injection analysis provides insight into why the execution model of a GPU, as described in Section III-A and shown in Section V-C for GEMM, will impact reliability when accelerating a CNN. As shown in Fig. 9, as errors propagate, they affect a larger percentage of data elements. Since 3-D matrices have different sizes at each layer, the absolute number of corrupted elements is not as insightful as the percentage of corrupted elements. Additionally, as YOLO performs 3-D convolutions on 3-D matrices, any corruption, while propagating, potentially affects a 3-D area. Recall that Fig. 7 showed how injections inside a layer manifest at the output of the layer. Most of those errors become cubic once digested by the downstream layers. Multiple corrupted elements at the output of GEMM are then likely to exacerbate the spread of errors in downstream layers. The higher is the percentage of corrupted elements, the lower is the probability for maxpooling to mask errors.

C. Reliable Maxpooling

Next, we reflect on our results to redesign the maxpool layers to detect, and possibly correct, faults. As shown in Fig. 9, errors spread quickly in a CNN. Injections leading to SDCs or critical SDCs tend to have all the output elements of the last layer corrupted. Alternatively, injections that are masked corrupt less than half of the elements of the CNN output. To make CNN execution more reliable, it is fundamental to detect errors promptly to prevent errors from propagating and from reaching the fully connected layers.

Additionally, we found that in a fault-free execution, the maximum absolute value for elements entering the maxpool layer, considering all of the frames in the Caltech and VOC datasets, is 21.15 for 1P layer, 12.23 for 3P layer, 8.39 for 8P layer, and 5.72 for 19P layer. The layer values are extremely small, considering the full range of floats or even integer or FP16) that can be represented, and that radiation can produce. Instead of simply propagating the element with the highest value, the maxpooling operation should also evaluate if the value of the element to propagate is *reasonable*.

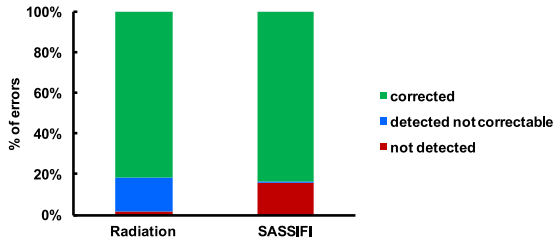


Fig. 10. Reliable maxpool effectiveness demonstrated in our radiation and fault-injection experiments.

A more reliable maxpool layer should evaluate if the value of the max element is greater than a threshold (to be conservative, we can set the threshold to be $10x$ the max value of a fault-free execution) and, if so, halt the processing of the current frame and move to the next frame. This solution will detect those faults in GPUs that affect multiple elements that, as demonstrated in Section V-C, are also the faults with the greatest impact on the final value. We can go a step further and correct errors by designing a maxpool layer that, when detecting a faulty max value, propagates the second greatest element, if *reasonable*. As the maxpool layer is intrinsically imprecise, propagating the second highest value does not significantly undermine the detection quality [67]. The overhead introduced to implement detection/correction is limited to four variables that hold the thresholds and the potential error detection overhead, which is done in O(1). This overhead is much lower than the overhead for ECC, which is O(logN).

In Fig. 10, we report the percentage of SDCs corrected, detected but not corrected (i.e., the second greatest element is not reasonable), and undetected with our maxpool layer. Data were obtained both with beam experiments and SASSIFI. Our solution corrects 81% of SDCs under the beam and 83% with SASSIFI. The most promising result is that our radiation experiments demonstrate that only 2% of the SDCs remain undetected, which is extremely close to the 99% detection limit ASIL-D imposes for self-driving vehicles.

With SASSIFI we can also track SDC propagation and we found that our maxpooling detects approximately 75% of all corrupted elements in the 1P layer, 84% in the 3P layer, 51% in the 8P layer, and 43% in the 19P layer—this assumes that these errors were not detected in upstream maxpool layers. Of those detected errors, 99.98% are corrected in 1P layer, 99.66% in 3P layer, 53.53% in 8P layer, and 48.64% in 19P layer. As the percentage of corrupted elements increases as errors propagate, it is harder for later maxpool layers to have a reasonable second greatest element to propagate.

VII. Conclusion

In this paper, we have evaluated the reliability of GPUs executing YOLO, Faster R-CNN, and ResNet, as run on three different NVIDIA GPUs exposed to atmosphericlike neutron beams. We have found that, for CNNs, crashes are more frequent than SDCs, and the higher number of operations executed in Faster R-CNN and ResNet makes them more prone to corruption than YOLO. We have also seen that transistor implementations can have a significant impact on the reliability of CNNs. The error rate of FinFET devices is found to be approximately an one order of magnitude lower than the error rate of CMOS devices. Additionally, GPU microarchitecture can propagate a single fault to affect several output elements, and this behavior significantly impacts CNN reliability.

Using Precision and Recall, we have distinguished between critical and tolerable errors in object-detection frameworks. Unfortunately, ECC is insufficient to ensure high reliability in CNNs, as it does not reduce the number of critical errors. We have found that alternative protection techniques, such as ABFT, can be much more effective in reducing the critical error rate.

We have also studied error propagation in YOLO using fault injection. Faults tend to spread during convolution, which suggests that to ensure high reliability faults should be promptly detected. We analyzed how faults propagate through GPUs when executing a CNN pipeline, and have designed maxpool layers that can detect radiation-induced errors at runtime.

In the future, we plan to evaluate model training to make CNN execution more reliable. New directions include placing normalization layers that remove detected corrupted data. Also, we plan to design watchdogs integrated in the GPU architecture to detect crashes and promptly restart computation to reduce the risk of missing deadlines in real-time systems. Finally, we plan to deeply investigate the impact of mixed precision on CNN reliability and efficiency.

Acknowledgement

This study was partially financed by the CoordenaÃ§Ã£o de AperfeiÃ§oamento de Pessoal de NÃvel Superior - Brasil (CAPES) - Finance Code 001. GPUs were donated by NVIDIA thanks to Siva Hari, Timothy Tsai, and Steve Keckler. Neutron beam time was provided by ChipIR (DOI: 10.5286/ISIS.E.87856107) and LANSCE thanks to Christopher Frost, Carlo Cazzaniga, and Stephen Wender.

References

[1] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, "LBANN: Livermore big artificial neural network HPC toolkit," in *Proc. Workshop Mach. Learn. High-Perform. Comput. Environ.*, ser. MLHPC '15. New York, NY, USA, 2015, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834897

[2] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[3] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, ser. MM '14. New York, NY, USA, 2014, pp. 675–678. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654889

[4] M. Bojarski *et al.*, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016, arXiv:1604.07316. [Online]. Available: http://arxiv.org/abs/1604.07316

[5] J. Dongarra, H. Meuer, and E. Strohmaier, "ISO26262 Standard, Road vehicles - Functional safety," 2015. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:dis:ed-2:v1:en

[6] J. C. Laprie, "Dependable computing and fault tolerance: Concepts and terminology," in *Proc. Highlights Twenty-Five Years. 25th Int. Symp. Fault-Tolerant Comput.*, Jun. 1995, pp. 2–11.

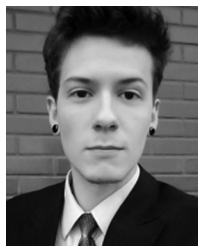
[7] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *Proc. IEEE 17th VLSI Test Symp.*, 1999, pp. 86–94.

[8] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 305–316, Sep. 2005.
[9] C. Alippi, V. Piuri, and M. Sami, "Sensitivity to errors in artificial neural networks: A behavioral approach," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 42, no. 6, pp. 358–361, Jun. 1995.
[10] S. Bettola and V. Piuri, "High performance fault-tolerant digital neural networks," *IEEE Trans. Comput.*, vol. 47, no. 3, pp. 357–363, Mar. 1998.
[11] UFRGS-CAROL, "Log database for IEEE transaction on reliability," *Github*. 2018. [Online]. Available: https://github.com/UFRGS-CAROL/transaction_on_reliability_2018
[12] K.-H. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, Jun. 1984.
[13] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Oct. 2017, pp. 249–258.
[14] V. Piuri, "Analysis of fault tolerance in artificial neural networks," *J. Parallel Distrib. Comput.*, vol. 61, no. 1, pp. 18–48, Jan. 2001. [Online]. Available: http://dx.doi.org/10.1006/jpdc.2000.1663
[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[16] G. Li *et al.*, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. ACM/IEEE Conf. Supercomput.*, 2017, pp. 1–11.
[17] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 446–456, Mar. 1995.
[18] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks," in *Proc. IJCNN Int. Joint Conf. Neural Netw.*, Jun. 1990, vol. 1, pp. 703–708.
[19] A. Kulakov, M. Zwolinski, and J. S. Reeve, "Fault tolerance in distributed neural computing," *CoRR*, vol. abs/1509.09199, 2015, arXiv:1509.09199. [Online]. Available: http://arxiv.org/abs/1509.09199
[20] S. Di Carlo, G. Gambardella, I. Martella, P. Prinetto, D. Rolfo, and P. Trotta, "Fault mitigation strategies for CUDA GPUs," in *Proc. IEEE Int. Test Conf.*, Sep. 2013, pp. 1–8.
[21] D. Oliveira *et al.*, "Radiation-induced error criticality in modern HPC parallel accelerators," in *Proc. IEEE 21st Symp. High Perform. Comput. Archit.*, 2017, pp. 577–588.
[22] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. Kaeli, "Calculating architectural vulnerability factors for spatial multi-bit transient faults," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Microarchit.*, Dec. 2014, pp. 293–305.
[23] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Mar. 2014, pp. 221–230.
[24] J. Tan, N. Goswami, T. Li, and X. Fu, "Analyzing soft-error vulnerability on GPGPU microarchitecture," in *Proc. IEEE Int. Symp. Workload Characterization*, Nov. 2011, pp. 226–235.
[25] N. DeBardeleben *et al.*, "GPU Behavior on a large HPC Cluster," in *Proc. 6th Workshop Resiliency High Perform. Comput. (Resilience) Clusters Clouds Grids Conjunction 19th Int. Eur. Conf. Parallel Distrib. Comput. (Euro-Par 2013)*, Aachen, Germany, Aug. 26–30, 2013, pp. 680–689.
[26] D. A. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 791–804, Mar. 2016.
[27] G. Li, K. Pattabiraman, C. Y. Cher, and P. Bose, "Understanding error propagation in GPGPU applications," in *Proc. SC16, Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Nov. 2016, pp. 240–251.
[28] M. Breuer, S. Gupta, and T. M. Mak, "Defect and error tolerance in the presence of massive numbers of defects," *IEEE Des. Test Comput.*, vol. 21, no. 3, pp. 216–227, May 2004.
[29] J. Wadden, A. Lyashevsky, S. Gurumurthi, V. Sridharan, and K. Skadron, "Real-world design and evaluation of compiler-managed GPU redundant multithreading," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, Jun. 2014, pp. 73–84.
[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 Trans. Nucl. Sci.*, vol. 60, no. 4, pp. 2797–2804, Aug. 2013.
[31] C. Braun, S. Halder, and H. J. Wunderlich, "A-ABFT: Autonomous algorithm-based fault tolerance for matrix multiplications on graphics processing units," in *Proc. IEEE/IFIP 44th Annu. Int. Conf. Dependable Syst. Netw.*, ser. DSN '14, Washington, DC, USA, 2014, pp. 443–454. [Online]. Available: http://dx.doi.org/10.1109/DSN.2014.48
[32] L. Pilla *et al.*, "Software-based hardening strategies for neutron sensitive FFT algorithms on GPUs," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 4, pp. 1874–1880, Aug. 2014.
[33] Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, JEDEC Standard, Tech. Rep. JESD89A, 2006.
[34] S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger, "Comparison of error rates in combinational and sequential logic," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 6, pp. 2209–2216, Dec. 1997.
[35] N. Mahatme *et al.*, "Comparison of combinational and sequential error rates for a deep submicron process," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 6, pp. 2719–2725, Dec. 2011.
[36] J. Noh *et al.*, "Study of neutron soft error rate (SER) sensitivity: Investigation of upset mechanisms by comparative simulation of FinFET and planar MOSFET SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 4, pp. 1642–1649, Aug. 2015.
[37] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3642–3649.
[38] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, ser. NIPS'12, 2012, pp. 2843–2851. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999325.2999452
[39] D. Ribeiro, A. Mateus, J. C. Nascimento, and P. Miraldo, "A real-time pedestrian detector using deep learning for human-aware navigation," *CoRR*, vol. abs/1607.04441, 2016, arXiv:1607.04436. [Online]. Available: http://arxiv.org/abs/1607.04441
[40] A. Angelova, A. Krizhevsky, V. Vanhoucke, A. Ogale, and D. Ferguson, "Real-time pedestrian detection with deep network cascades," in *Proc. Brit. Mach. Vis. Conf.*, 2015, pp. 4–16.
[41] S. Chetlur *et al.*, "cuDNN: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014, arXiv:1410.0759. [Online]. Available: http://arxiv.org/abs/1410.0759
[42] T. Luo *et al.*, "DaDianNao: A neural network supercomputer," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 73–88, Jan. 2017.
[43] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, ser. ISCA '17, New York, NY, USA, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246
[44] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[45] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *Comput. Vis. – ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Ed. Cham: Springer International Publishing, pp. 630–645, 2016, arXiv:1603.05027. [Online]. Available: http://arxiv.org/abs/1603.05027
[46] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," *BigLearn, NIPS Workshop*, 2011. [Online]. Available: http://publications.idiap.ch/downloads/papers/2011/Collobert_NIPSWORKSHOP_2011.pdf
[47] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.
[48] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11263-009-0275-4
[49] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 304–311.
[50] S.-H. Kim *et al.*, "A low power and highly reliable 400Mbps mobile DDR SDRAM with on-chip distributed ECC," in *Proc. IEEE Asian Solid-State Circuits Conf.*, Nov. 2007, pp. 34–37.
[51] H. M. Chen *et al.*, "Using low cost erasure and error correction schemes to improve reliability of commodity DRAM systems," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3766–3779, Dec. 2016.
[52] G. Reis, J. Chang, N. Vachharajani, and S. Mukherjee, "Design and evaluation of hybrid fault-detection systems," in *Proc. Int. Symp. Comput. Archit.*, 2005, pp. 148–159.
[53] L. L. Pilla *et al.*, "Software-based hardening strategies for neutron sensitive FFT algorithms on GPUs," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 4, pp. 1874–1880, Aug. 2014.

[54] Y. P. Fang and A. S. Oates, "Characterization of single bit and multiple cell soft error events in planar and FinFET SRAMs," *IEEE Trans. Device Mater. Rel.*, vol. 16, no. 2, pp. 132–137, Jun. 2016.
[55] D. Tiwari *et al.*, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *Proc. IEEE 21st Symp. High Perform. Comput. Archit.*, 2015, pp. 331–342.
[56] P. W. Protzel, D. L. Palumbo, and M. K. Arras, "Performance and fault-tolerance of neural networks for optimization," *IEEE Trans. Neural Netw.*, vol. 4, no. 4, pp. 600–614, Jul. 1993.
[57] E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak, "Investigating the fault tolerance of neural networks," *Neural Comput.*, vol. 17, no. 7, pp. 1646–1664, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1162/0899766053723096
[58] K. Pattabiraman, G. P. Saggese, D. Chen, Z. Kalbarczyk, and R. K. Iyer, "Dynamic derivation of application-specific error detectors and their implementation in hardware," in *Proc. 6th Eur. Dependable Comput. Conf.*, Oct. 2006, pp. 97–108.
[59] G. Candea and A. Fox, "Recursive restartability: Turning the reboot sledgehammer into a scalpel," in *Proc. 8th Workshop Hot Top. Oper. Syst.*, ser. HOTOS '01. Washington, DC, USA, 2001, pp. 125–. [Online]. Available: http://dl.acm.org/citation.cfm?id=874075.876408
[60] X. Wang, E. Türetken, F. Fleuret, and P. Fua, *Tracking Interacting Objects Optimally Using Integer Programming*. New York, NY, USA: Springer-Verlag, 2014, pp. 17–32. [Online]. Available: https://doi.org/10.1007/978-3-319-10590-1_2
[61] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2014, pp. 1–6.
[62] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Apr. 2015, pp. 1131–1135.
[63] P. Rech, C. Frost, and L. Carro, "GPUs neutron sensitivity dependence on data type," *J. Electron. Test.*, vol. 30, no. 3, pp. 307–316, 2014.
[64] V. Sridharan and D. R. Kaeli, "The effect of input data on program vulnerability," in *Proc. Workshop Silicon Errors Logic Syst. Eff.*, ser. SELSE '09, 2009, pp. 1–6.
[65] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proc. IEEE/ACM 36th Annu. Int. Symp. Microarchit.*, Washington, DC, USA, 2003, pp. 29–40.
[66] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proc. Des. Automat. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 502–506.
[67] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition" in *Proc. 20th Int. Conf. Artif. Neural Netw.*, 2010, pp. 92–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15825-4_10

Fernando Fernandes dos Santos received the B.Sc. degree in computer science from State University of Western Paran (UNIOESTE), Cascavel, Brazil, and the M.Sc. degree in computer science from Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2014 and 2016, respectively. He is currently working toward the Ph.D. degree at UFRGS, working on fault tolerance in embedded and safety-critical applications.

Pedro Foletto Pimenta has been working toward the Graduate degree in computer science at Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, since 2014. He is also taking part in an exchange program, studying at CPE Lyon, Lyon, France.



Caio Lunardi is currently working toward the Undergraduate degree in computer engineering at Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil. He is currently working on fault tolerance in HPC systems.

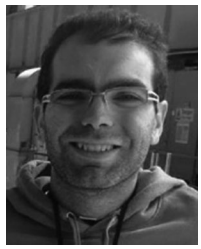
Lucas Draghetti is currently working toward the Undergraduate degree at Universidade Federal do Rio Grande do Sul(UFRGS), Porto Alegre, Brazil. He is currently working on fault tolerance in HPC systems.

Luigi Carro (M'86) received the Electrical Engineering and the M.Sc. degrees from Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1985 and 1989, respectively. He received the Dr. degree in computer science from UFRGS in 1996.

He is currently a Full Professor of computer architecture and organization with the Department of Applied Informatics, Informatics Institute, UFRGS. He has advised more than 20 graduate students, and has authored or coauthored more than 150 technical papers on those topics.

David Kaeli (M'09) received the B.S. and Ph.D. degrees in electrical engineering from Rutgers University, New Brunswick, NJ, USA, and the M.S. degree in computer engineering from Syracuse University, Syracuse, NY, USA.

He is currently a Distinguished Full Professor with the ECE faculty, Northeastern University, Boston, MA, USA. He has authored or coauthored more than 300 critically reviewed publications, 7 books, and 13 patents. His research interests include microarchitecture to back-end compilers, high performance computing, cybersecurity, reliability, and big data applications.



Paolo Rech (M'15) received the Master's degree in in computer engineering and Ph.D. degree in computer engineering: information science, and technology from Padova University, Padova, Italy, in 2006 and 2010, respectively.

He is an Associate Professor with Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil. His main research interests include the reliability of FPGAs, SoC, parallel and heterogeneous systems; the design of efficient hardening techniques, and the evaluation and mitigation of radiation-induced effects in devices designed for large-scale HPC centers and safety-critical applications.