HAL open science

Characterizing a Neutron-Induced Fault Model for Deep Neural Networks

Fernando Fernandes dos Santos, Angeliki Kritikakou, Josie Esteban Rodriguez Condia, Juan David GUERRERO BALAGUERA, Matteo Sonza Reorda, Olivier Sentieys, Paolo Rech

To cite this version:

Fernando Fernandes dos Santos, Angeliki Kritikakou, Josie Esteban Rodriguez Condia, Juan David GUERRERO BALAGUERA, Matteo Sonza Reorda, et al.. Characterizing a Neutron-Induced Fault Model for Deep Neural Networks. IEEE Transactions on Nuclear Science, In press. hal-03865253v1

HAL Id: hal-03865253

https://inria.hal.science/hal-03865253v1

Submitted on 22 Nov 2022 (v1), last revised 29 Nov 2022 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Characterizing a Neutron-Induced Fault Model for Deep Neural Networks

Fernando Fernandes dos Santos, Angeliki Kritikakou, Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera
Matteo Sonza Reorda, Olivier Sentieys, and Paolo Rech

***Abstract*—The reliability evaluation of Deep Neural Networks (DNNs) executed on Graphic Processing Units (GPUs) is a challenging problem since the hardware architecture is highly complex and the software frameworks are composed of many layers of abstraction. While software-level fault injection is a common and fast way to evaluate the reliability of complex applications, it may produce unrealistic results since it has limited access to the hardware resources and the adopted fault models may be too naive (i.e., single and double bit flip). Contrarily, physical fault injection with neutron beam provides realistic error rates but lacks fault propagation visibility. This paper proposes a characterization of the DNN fault model combining both neutron beam experiments and fault injection at software level. We exposed GPUs running General Matrix Multiplication (GEMM) and DNNs to beam neutrons to measure their error rate. On DNNs, we observe that the percentage of critical errors can be up to 61%, and show that ECC is ineffective in reducing critical errors. We then performed a complementary software-level fault injection, using fault models derived from RTL simulations. Our results show that by injecting complex fault models, the YOLOv3 misdetection rate is validated to be very close to the rate measured with beam experiments, which is 8.66× higher than the one measured with fault injection using only single-bit flips.**

I. Introduction

Deep neural Networks (DNNs) are efficient tools to perform several tasks, such as object detection, image segmentation, classification, and prediction [1]–[3]. DNNs have been deployed in safety-critical and mission-critical domains, such as robotics, aeronautics and space exploration, smart healthcare, and autonomous driving. To reduce the processing time and energy consumption of DNN implementations, dedicated architectures with specialized hardware or reduced precision, e.g., half-precision floating-point (16 bits) or even short integer (8 bits), have been proposed, with satisfactory classification, segmentation, and detection accuracy. As an example, Modern Graphics Processing Units (GPUs) feature Tensor Cores, i.e., a dedicated hardware to perform 4×4 matrix multiplications with a single instruction [4]. Also, for specific domains, ASIC accelerators are employed for DNN, achieving significant performance and low power consumption [5].

When DNNs are deployed in safety-critical applications, real-time execution and reliability are essential. Thus, to leverage the benefits of DNNs towards safety-critical systems, a DNN-based system must be compliant with standards, such as ISO 26262 and ISO/PAS 21448. However, system reliability can be undermined by several sources, including environmental perturbations, aging, and process/temperature/voltage variations [6], [7]. Radiation-induced soft errors are particularly critical, as they dominate error rates in commercial devices [8] and are very challenging due to their random and transient nature [9]. As a consequence, DNNs are susceptible to transient faults induced by radiation [10]–[14]. Additionally, GPUs have a high fault rate because of the high amount of available resources [15]–[17] and the possibility to have multiple output elements corrupted, which undermines DNN reliability [10], [12].

The reliability evaluation of GPU architectures is then mandatory to characterize the errors that can hinder the DNN accuracy. A realistic method to evaluate the reliability of DNNs on GPUs is physical-level fault injection with a neutron beam. However, with beam experiments, only the fault outcome in the application's output can be observed. Thus, fault propagation visibility is missing. Contrarily, fault injection at the software level has limited overhead, making it possible to perform several fault injection campaigns in order to identify the faults that impact the DNN reliability. Despite being fast, software-level fault injection is restricted by the resources where faults can be injected, such as register file and variables, limiting the accuracy of the fault models. The main goal of this work is to bridge this gap by obtaining a realistic DNNs' transient error model, extracted from neutron beam experiments, and compare it with different error models, obtained from software-level fault injections.

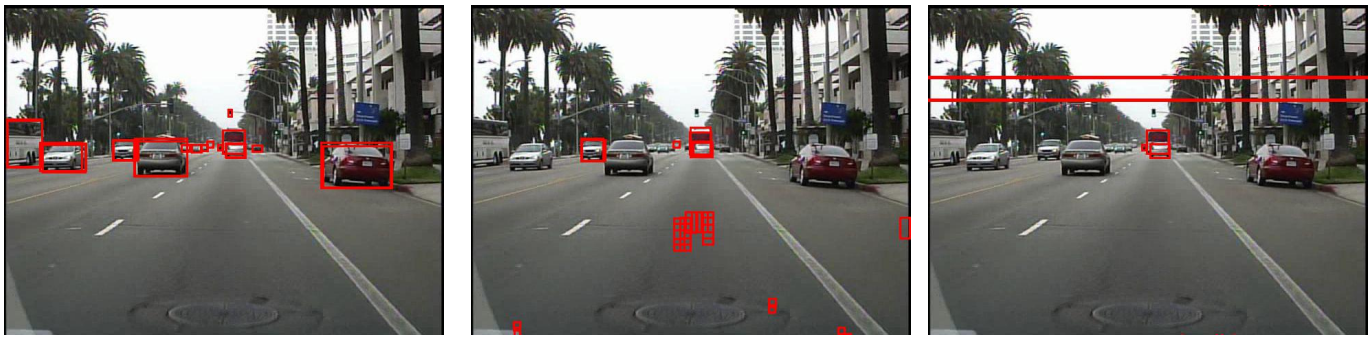
We report findings from beam testing campaigns that assess GPU reliability as a DNN accelerator and identify the causes of *critical* errors (i.e., errors that impact detection) and *tolerable* errors (i.e., errors that do not impact the detection). Our testing campaigns consider two different GPU architectures, i.e., Kepler (Tesla K40) and Volta (Tesla V100), and several configurations for multiple DNNs (YOLO, Faster R-CNN, ResNet). These configurations include different floating-point precisions and the use or non-use of specialized cores and of Error Correction Codes (ECC). We also evaluated the error rate of the fundamental operations of DNNs, the General Matrix Multiplication (GEMM) kernel. The results show that a single

Fernando Fernandes dos Santos, Angeliki Kritikakou, and Olivier Sentieys are with University of Rennes, INRIA, France. E-mail: fernando.fernandes-dos-santos@inria.fr, angeliki.kritikakou@irisa.fr, and olivier.sentieys@inria.fr.

Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera, and Matteo Sonza Reorda are with Politecnico di Torino - Department of Control and Computer Engineering DAUIN. E-mail: {josie.rodriguez, juan.guerrero, matteo.sonzareorda}@polito.it

P. Rech is with University of Trento - Department of Industrial Engineering. E-mail: paolo.rech@unitn.it

978-1-5386-5541-2/18/$31.00 ©2018 IEEE



(a) Expected object detection (b) Critical SDC from radiation experiments (c) Critical SDC from NVBitFI fault injection

Fig. 1: Figure 1a shows the expected outcome from the YOLOv3 DNN when no fault is observed. Figures 1b and 1c show the object detection when a fault perturbates the execution of the DNN from neutron beam experiments and software-level fault injection, respectively. Both Figures 1b and 1c are Critical SDCs. The fault model that generated Figure 1c is based on the proposed complex fault model. The frame is extracted from Caltech Pedestrian detection dataset [18].

particle can spread through the GPU microarchitecture, affecting several parallel threads and output elements, significantly impacting DNN reliability. Furthermore, for DNNs, Single Error Correction Double Error Detection (SECDED) ECC can be useful, but not always effective. Although the ECC can reduce the error rate by an order of magnitude, ECC can reduce neither the number of critical nor multiple errors that come from unprotected GPU units. We show that even with ECC protection, the critical error rate is not neglectable.

We modify the NVIDIA Binary Instrumentation Fault Injector (NVBitFI) [19] to evaluate the fault propagation for complex fault models on GEMM and DNNs. With NVBitFI, we are able to mimic the critical errors observed in beam experiments. We inject 2,000 faults on YOLOv3 DNN and 644 faults on GEMM per fault model for each configuration, accounting for more than 35,000 injected faults. The contributions of this paper are as follows:

- As we perform the experiments with an accelerated neutron beam (with a flux $\approx 8\times$ higher than the terrestrial flux), we provide a realistic error rate that accounts for more than 380,000 years of terrestrial device operation. We show the results for two different GPU architectures (Kepler and Volta), with different floating-point precisions (FP32 and FP16).
- We measure the efficacy of the ECC in protecting the DNNs against errors that impact the classification/detection. We also characterize the criticality of the errors that modify the GEMM output when used as the core of the DNNs operations.
- We present a complete characterization of the fault model at the software level on GEMM and DNNs. On this fault propagation evaluation, we can identify which types of simulated faults affect the DNN output similarly to the beam experiments.
- We craft an improved version of NVBitFi that includes multiple thread fault models and compare them with the standard single-bit flip.

The rest of this paper is organised as follows. Section II presents the background on radiation-induced errors on DNNs. Section III introduces the proposed analysis and shows the experimental methodology for the beam experiments and the fault simulation at the software level. The GEMM and DNNs error rate is presented in Section IV. Section V presents the fault injection results at the software level using the proposed fault models on YOLOv3 with FP16 and FP32. Finally, Section VI concludes the paper.

II. Radiation induced errors in DNNs

Terrestrial neutrons can interact with electronic devices, and a transistor's state can be perturbed by a neutron strike (Single Event Transient, SET), thus generating bit-flips in memory or current spikes in logic circuits that, if latched, may lead to an error [20] (i.e., Single Event Upset, SEU). An error generated by a neutron is called a soft error, as most of the time, it does not damage the device, but it can change the output of an application (e.g., a DNN classification). On GPUs, as with any other device, the fault propagates from the hardware to the software level leading to the possible outcomes:

1) **Masked**: no effect on the program output. The corrupted data is not used, or the circuit functionality is not affected.
2) **Detected Unrecoverable Error (DUE)**: the program stops working or the entire system crashes.
3) **Silent Data Corruption (SDC)**: undetected output corruption, that is, the application finishes, but the application output is not correct and no flag or indication of an error is set.

The SDCs on DNNs have a particular characteristic compared to other typical GPU applications. The SDCs can be separated into two classes: (1) **Tolerable SDCs** are the errors that change the output of the last layer of the DNN, but they do not modify the classification or object detection. (2) **Critical SDCs** are the errors that change the classification or detection partially or entirely. The Critical SDCs are particularly dangerous for safety-critical applications, as a decision based on an incorrect inference can lead to a catastrophic event. For instance, Figure 1 shows the result based on a real scenario extracted from Caltech Pedestrian detection dataset [18]. The expected object detection is shown

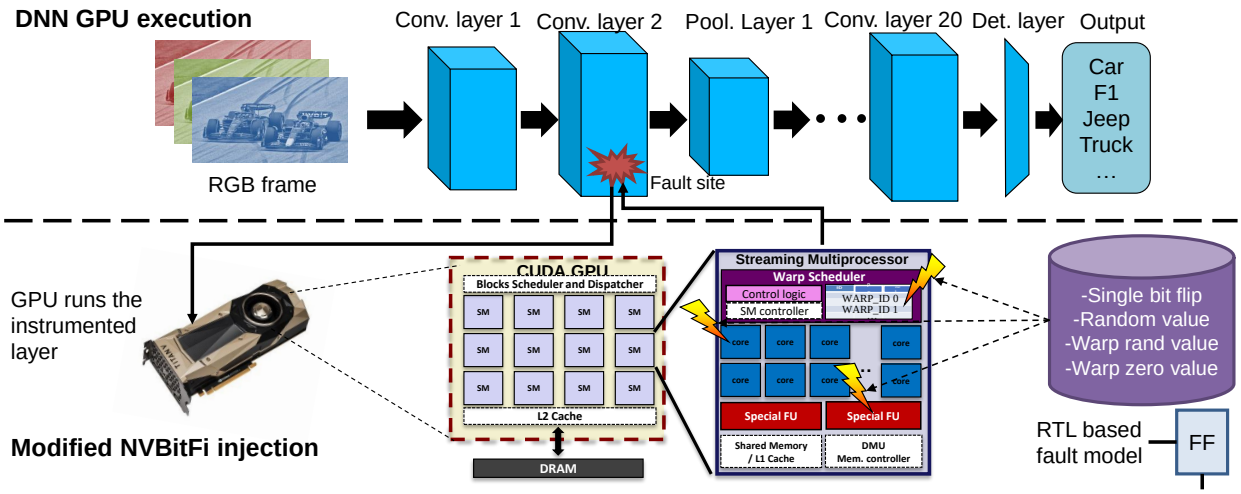


Fig. 2: Proposed software-level fault approach. The DNN receives an RGB frame in the first convolution and propagates it into the subsequent layers. In the modified version of the NVBitFi, the kernels are instrumented, and a random fault site inside the layer kernel is selected. Using the fault models based on [21], we can inject multiple types of faults, corrupting single registers or one register in an entire GPU warp.

in Figure 1a. Figures 1b and 1c show two examples of Critical SDCs observed on neutron beam experiments and software-level fault injection, respectively. In fact, critical SDCs can be detected at the software level with different types of fault tolerance, such as temporal redundancy [22], [23], algorithm-based fault tolerance [10], [12], [17], [24], [25], and float point value clipping [10], [26]. However, all the software-level approaches add overhead to the DNN inference. Contrarily, we propose a better characterization of the fault model of the DNNs. Then researchers can use more efficient ways to detect faults, such as fault-aware training to harden the DNN from the design phase, which is only possible by knowing the correct fault model of the accelerator beforehand.

Different types of fault injection are available to evaluate the reliability of GPUs, such as physical-level fault injection with beam experiments and fault simulation at the software level. In previous works, the GPU error rate for typical applications has been evaluated through beam experiments where the device is exposed to a particle flux and the error rate is measured [10], [17], [27]–[30]. When the accelerated particle beam goes through the hardware, it induces transient faults, which may lead to output errors. The main characteristic of neutron beam experiments is that the whole device is irradiated, and thus, faults are not injected only to a subset of resources. To measure the neutron-induced error rate of DNNs, researchers exposed GPUs [10], [12], FPGAs [31], and ASIC accelerators [14], running various types of DNNs. Beam experiments provide a realistic error rate of the device running a code. However, we cannot distinguish which level of the hardware or the software contributes more to the device error rate, making it challenging to identify the most vulnerable parts of the system. Knowledge of fault propagation through the different hardware and software layers would help system designers to improve reliability more effectively.

In order to understand the faults' propagation, fault injection is performed by simulation. Faults can be injected at different levels of abstraction, from low-level Register-Transfer Level (RTL) [32]–[34] to microarchitecture [35], [36] and software [19], [37]–[41]. Each level of abstraction provides the propagation probability of the injected fault to the output, measured in terms of Hardware, Architectural, Software, or Program Vulnerability Factor (HVF, AVF, SVF, PVF, respectively) [42], [43]. Fault injection assumes that the fault already occurred and provides the fault propagation through the system. However, this approach has limitations, since the fault model and fault injection probabilities are defined by the user and the simulator, which can lead to unrealistic results. Moreover, faults can be injected only into the available and accessible resources, which is a subset of the resources and not the entire system, as with beam experiments.

In recent works, the DNNs fault model on GPUs has been analyzed to provide a more realistic reliability evaluation using fault injection [10], [21], [41]. The fault model analysis is based on the key concept of matrix multiplications for DNNs. Most DNN layers, such as convolutional, recurrent, and fully connected, can be represented as General Matrix Multiplication (GEMM). Then, the reliability evaluation can be performed at the GEMM module level. Consequently, the fault impact on the GEMM output is limited to a 2D geometry format. Therefore, once the fault is injected, the error will manifest as a corruption of a single element, line/row, or block of the output matrix. The main limitation of the recent works regarding realistic fault models is that they are either based on beam experiments, or they perform fault injection in a high level of abstraction (C++/Python), without taking into account the GPU architecture details that impact the fault propagation.

In this work, we leverage the knowledge obtained in the past works with beam experiments [10] and RTL fault injections [21] to propose an analysis that considers the fault propagation in GEMM modules with Shader Assembler (SASS) -level fault injections (i.e., assembly level on NVIDIA GPUs).

III. Evaluation Methodology

This section presents the analysis we propose to characterize the GPU fault model for DNNs and how we modify the NVBitFI to meet our needs. Then, we carefully present the characterized GPU devices, the considered DNNs, the metrics adopted for the reliability evaluation, and how the beam experiments and fault injections are performed.

A. Proposed analysis

For the first time, we propose implementing and evaluating different fault models for GPUs at the instruction-level. The proposed fault models are based on the fault syndrome observed on RTL fault injection [21], where the authors reported multiple fault syndromes based on fault injections on a synthetic RTL version of an NVIDIA microarchitecture GPU [34], [44]. We inject faults on DNNs using an instruction-level fault injector (NVBitFI). With more realistic fault models, we can verify whether the DNN criticality obtained from fault simulation is comparable to the one observed with beam experiments. In fact, the warp-wide fault models were previously proposed by Siva *et al.* [45] in the SASSFI fault injector. However, SASSIFI only supports older NVIDIA architectures, making it impossible to simulate complex applications like DNNs on newer GPU architectures. Additionally, our analysis is not limited to the fault model itself, but also considers the consequences of complex faults on the DNNs inference process and compares them with the results from beam experiments.

Figure 2 shows the proposed SASS level fault injection methodology. When a GPU kernel is launched, the threads are split into small groups of 32 threads called *Warps*. A Warp is the smallest unit of execution on NVIDIA GPUs, where all threads within a Warp start executing the kernel instructions together and finish simultaneously. As reported in past works [21], [27]–[29], [34], [46], the Warp scheduler and the control logic, responsible for managing the Warp's threads, are critical resources, where a single event upset can generate errors affecting all the 32 Warp threads. Aiming to model this behavior at the software level, we modify the NVBitFI fault injector to support two new fault models, i.e., Warp random value and Warp zero value. These two fault models are derived from RTL level fault injections [21], where a single fault on GPU shared resources, such as Warp scheduler and special functional units, is capable of affecting the output of all threads in a GPU Warp.

The modified version of NVBitFI selects a random fault site on the DNN under evaluation. The fault site can be any module of the DNN. Then, we select one of the following fault models to be injected into the output of one or more instruction register(s): (1) standard **Single Bit Flip** in the output of one instruction (32-bit register); (2) a **Random value** to replace the value in the instruction output; (3) **Warp random value**, where all the threads within a warp will have at least one instruction output replaced by a random value; (4) **Warp zero value**, which is similar to Warp random value, but we replace with a zero value instead of a random one.

B. GPU devices

We consider Kepler (Tesla K40) and Volta (Tesla V100) NVIDIA GPUs. The tested NVIDIA K40 (**Kepler**) is built with the Kepler ISA and fabricated in a $28nm$ TSMC standard CMOS technology [47]. Tesla V100 (**Volta**) are designed with the Volta microarchitecture and built with TSMC FinFET $12nm$ [4]. Volta GPUs feature hardware acceleration for three IEEE 754-2008 floating-point point precisions: double (FP64), single (FP32), and half (FP16). We also evaluate the reliability of Volta GPUs' *tensor cores*, i.e., a specific hardware that performs 4×4 Matrix Multiplication from GEMM kernels in one cycle with FP16 precision. Tensor cores can also perform matrix multiplication for FP32 precision, however, the data will be cast to FP16 in the low-level operation. Both GPU architectures have available Single Error Correction Double Error Detection (SECDED) Error Correcting Code (ECC) to protect the register file, shared memory, and caches. Our evaluation considers errors occurring only in the GPU core, not in the main memory. For Kepler devices, we chose a beam spot that is sufficiently small (2cm of diameter) in order to not hit the onboard DRAM, when ECC is disabled. Since Volta GPUs' DRAM is too close to the GPU core, we choose to test Volta GPUs only with ECC enabled.

C. Deep Neural Networks

This work considers three modern DNN models: i) You Only Look Once (YOLO v1 and v3) [48], [49], ii) a Faster Region-based Convolution Neural Network (Faster R-CNN) [2], and iii) a Residual Network (Resnet) [1]. **YOLO** is based on *Darknet*, which is an open-source CNN in C and CUDA [48], [49]. **Faster R-CNN** is written in C++ and Python, based on the Caffe [50] deep learning framework. **ResNet** is a CNN based on the Torch deep learning framework [51]. ResNet only performs object classification, while YOLO and Faster R-CNN also provide object detection.

We classify the observed SDCs between Tolerable SDCs (i.e., that do not impact classification/detection) and Critical SDCs (i.e., that impact classification/detection). When radiation modifies the classification or detection, we check whether all the objects are detected and classified correctly. Then, we check if the error created a false positive (i.e., add non-existing objects) or if the error made the DNN misdetect objects. For the beam experiments and the fault injection, we evaluate the reliability of the DNNs with two datasets: Caltech Pedestrian detection [18] and VOC2012 [52]. Note that, evaluating with fault injection all the frames tested on neutron experiments would be unfeasible as the fault sites grow exponentially. Thus, to reduce the fault space for the SASS-level fault injections, we select the frames from the Caltech dataset that generated critical SDCs on the beam experiments.

D. Beam Experiment Setup

Our experiments were performed at the ChipIR facility of the Rutherford Appleton Laboratory, UK, and at the LANSCE facility at Los Alamos National Laboratory, US. Figure 3 shows the setup mounted in the ChipIR facility. Both facilities

deliver a beam of neutrons with a spectrum of energies similar to the atmospheric neutron one [53]. The available neutron flux was about $3.5 \times 10^6 n/(cm^2/s)$, ~8 orders of magnitude higher than the terrestrial flux (13 $neutrons/(cm^2 \cdot h)$ at sea level [54]). The Failure In Time (FIT) rate is calculated by dividing the number of observed errors by the received particles fluence ($neutrons/cm^2$).

Since the terrestrial neutron flux is low, it is improbable to see more than a single corruption during program execution in a realistic application. We have carefully designed the experiments to maintain this property (observed error rates were lower than 1 error per 1,000 executions). In this way, the experimental data can be scaled to the natural terrestrial environment without introducing artifacts. Each DNN was tested for at least 24 effective hours, not including the setup, result check, initialization, and recovery from the DUE time.

We added setup software and hardware watchdogs to monitor the experiments (Figure 3). The software watchdog controls the application under test, and if it stops responding in a predefined time interval, the kernel is killed and relaunched. This watchdog detects kernel crashes or software hangs, i.e., application crashes or control flow errors occur that prevent the GPU from completing assigned tasks (e.g., an infinite loop). The hardware watchdog is an Ethernet-controlled switch that performs a host computer's power cycle, if the host computer itself does not acknowledge any ping requests in a predefined time interval. The hardware watchdog is necessary to detect operating system hangs. The source code of our setup and other research artifacts used in this paper are available in online repositories (details in Appendix A).

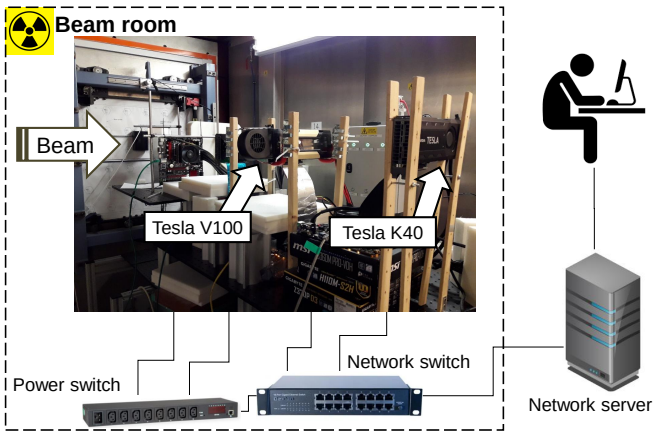


Fig. 3: Experimental setup at ChipIR at Ruthenford Appleton Laboratory in UK. Boards are placed on the beam room and exposed to a neutron flux of $\approx 3.5 \times 10^6 n/(cm^2/s)$. The network server is composed of a set of Python scripts which controls the devices through the Ethernet and performs power cycles through a power switch.

E. SASS-level fault injection

To perform the SASS-level fault injection, we have updated the available NVBitFI framework [19]. NVBitFI allows instrumenting the GPU kernels at the assembly level (i.e., at the microinstruction level). Other fault injectors, such as GPUQin, CAROL-FI, Kayotee, GPGPU-SIM, SASSIFI [38], [45], [55], [56] neither inject at the SASS level nor offer support for Volta and newer architectures, nor inject in CUDA libraries.

NVBitFI can inject transient errors in the GPU's ISA visible states, modifying the SASS instructions output of a code being executed on a real GPU. By default, NVBitFI allows for the users to select the fault model to inject single and double bit-flip, single zero value, and a single random value. We modify the injection procedure to inject the multiple thread faults, Warp random value and Warp zero value. As we used an NVIDIA CUDA Warp primitive to perform the synchronization between the Warp threads, our modifications add a low overhead to the expected NVBitFI injection overhead. Our modified NVBitFI is similar to the original version, but when the Warp-wide injection mode is selected, it will modify one output of one floating-point instruction in all threads within a random warp of the GPU.

IV. DNNs Error Rate

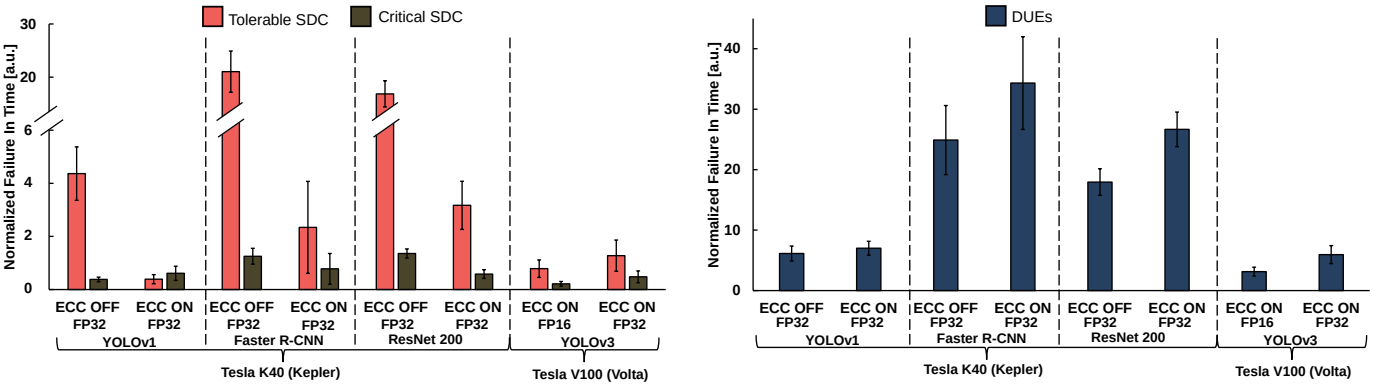
This section presents the results from radiation experiments for the evaluated Deep Neural Networks (DNNs). To provide insight into the DNNs' error model, we present an evaluation of the error rate and model of GEMM, which is the core of current DNNs.

A. Deep Neural Networks FIT rate

Figures 4a and 4b show the normalized SDC and DUE rates, respectively. We show results for YOLOv1, Faster R-CNN, and ResNet on Kepler and YOLOv3 on Volta GPUs. We separate the Tolerable and Critical SDCs. The reported results are normalized to not reveal business-sensitive information. Even if the data is normalized, it allows a direct comparison among configurations. The reported values are relative to YOLOv1 ECC ON SDC FIT for Kepler and YOLOv3 ECC ON SDC FIT for Volta. Experimental data is presented with a 95% confidence interval.

Figure 4 shows that DUEs are more probable than SDCs for all tested configurations. DNN kernels have a high level of reuse that requires several device-host (CPU-GPU) synchronizations. A transient fault during these synchronizations could potentially result in a GPU DUE. As a result, while a significant portion of SDCs could be masked, DUEs could still undermine the device's reliability. For the same reason, Faster R-CNN and Resnet, which require a much larger number of synchronizations, show up to 5× higher DUE rate than YOLO.

The impact of the ECC on the GPU's DUE rate has been evaluated in past works [16], [46], which show that the DUE rate increases when ECC is ON, particularly in the case of memory-intensive codes. With ECC enabled, the DUE rate increases by up to 30% for Faster R-CNN, Resnet, and YOLOv1. ECC is able to correct one-bit flip in the protected memories, and, when a double-bit flip is detected, it throws a system exception. Consequently, as DNNs use a large memory to perform classification/detection, multiple errors on memories are expected to happen, resulting in a higher DUE rate when ECC is ON. Equivalently, on Volta GPU, the DUE rate grows as the data representation increases. Comparing



(a) (b)

Fig. 4: Figure 4a and 4b show the normalized Failure In Time of SDC and DUE, respectively, for YOLOv1, Faster R-CNN, ResNet, and YOLOv3. The SDC rate is separated between Tolerable errors (i.e., the inference is not modified) and Critical SDCs (i.e., the inference is modified).

FP16 and FP32 versions of YOLOv3, the memory size grows $2\times$. As the FP32 version of YOLOv3 performs more memory transfers than the FP16 version, the DUE is expected to be higher for FP32. Additionally, with the ECC ON on Volta, and when caches and register usage increases to store the FP32 precision, the double bit flips in the memories are more frequent.

Figure 4a shows that the SDC rates are related to the DNN model complexity and accuracy. Compared to YOLO, the complex structures used on Faster R-CNN and Resnet increase their SDC rate by more than $10\times$. Figure 4a shows that Resnet has a higher FIT rate than YOLO (i.e., v1 and v3), though the rate is similar to Faster R-CNN. Although Faster R-CNN and ResNet have a high detection/classification accuracy, it is insufficient to compensate for the higher error rate associated with the complex framework used for both DNNs. A more accurate DNN, most of the time, demands more computations to be performed. Consequently, as the error rate is directly proportional to the amount of resources used, the more resource demanded, the higher the error rate.

The SDC rate for YOLOv1, Faster R-CNN, and Resnet with ECC ON is 21%, 13.6%, and 22% less than the SDC rate seen with ECC OFF, respectively. ECC is not as effective in these workloads as in other codes, mainly because neural networks are intrinsically resilient to data errors. ECC can reduce the GEMM SDC rate by about one order of magnitude [27] (details in Section IV-B). However, ECC is less efficient in protecting DNNs, as some of the SDCs that ECC masks would not have affected the DNN execution. Figure 4a shows that even if ECC significantly reduces the number of Tolerable SDCs, ECC fails at reducing the occurrence of Critical SDCs. On Volta GPU, YOLOv3 FP16 has a lower FIT than YOLOv3 FP32. The amount of per-core resources required to perform the operations, depends on the chosen data precision. Namely, fewer resources will be used for the lower floating-point precision.

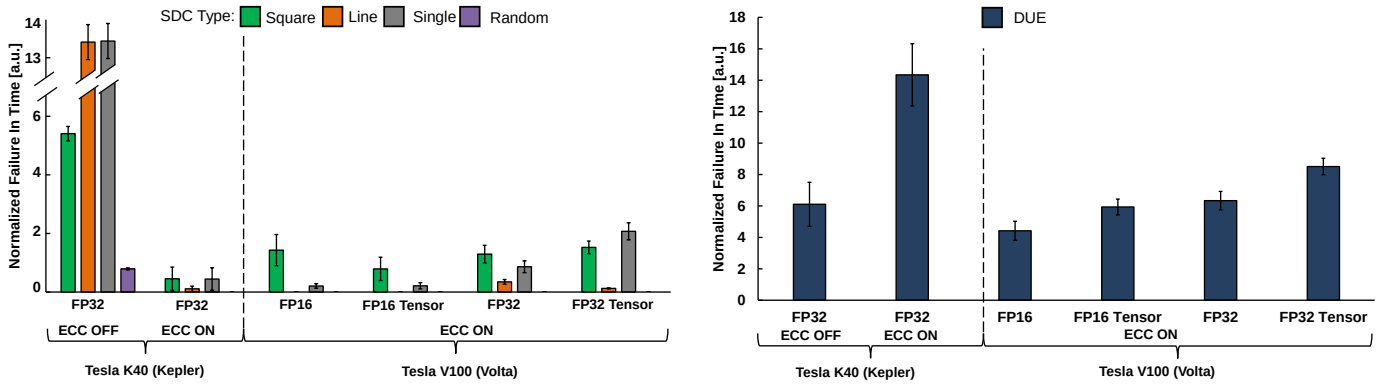
For object detection, the percentage of critical SDCs is much lower for Faster R-CNN and YOLOv3 than for YOLOv1. For YOLOv1, the percentage of critical SDCs is 8% with ECC OFF and 61% for the Kepler with ECC ON. For YOLOv3 with ECC ON, the percentage of critical SDCs is 21% and 27% for FP16 and FP32, respectively. For Faster R-CNN, the critical SDCs are 5% with ECC OFF and 25% for the Kepler with ECC ON. The difference in the percentages of critical errors among DNNs comes from the detection mechanism. YOLOv1 is the simplest DNN tested in our setup with 31 layers. An error is expected to impact much more the detection of simpler DNNs than more complex ones.

Although ECC reduces the error rate, the portion of critical errors is not reduced at the same rate as tolerable errors. This can be explained with the fine-grain analysis based on GPU-Qin [26]. As shown in [26], ECC does not mask all possible faults but only the ones occurring in the protected memories, while faults in computing elements can propagate to the output. ECC reduces the absolute number of SDCs, but it has the side effect of increasing the portion of multiple errors (details in Section IV-B), which are more likely to propagate through DNN layers and affect detection. On Volta, the percentages of critical errors are similar comparing the two data types when ECC is ON. It is worth noting that critical SDCs are less dependent on data type as the two DNNs have similar detection accuracy regarding the data precision.

B. Complex error models for DNNs

To completely characterize the critical errors on DNNs and how they can be generated at low level, we evaluated how the errors affect the General Matrix Multiplication algorithm (GEMM), used as the core of DNNs execution on modern GPUs. In fact, 70% to 86% of the operations for the evaluated DNNs are GEMM related operations.

Figure 5 shows the SDC and DUE normalized FIT rates for GEMM, respectively. Kepler values are relative to the SDC rate with ECC enabled, while Volta values are relative to the SDC rate of FP16 without Tensor cores. We selected matrix sizes that saturate the resources of each device (2048×2048 for the Kepler and 16384×16384 for Tesla V100). As device



(a) (b)

Fig. 5: Figure 5a shows the normalized SDC Failure In Time (FiT) for the General Matrix Multiplication (GEMM) for Kepler and Volta. The SDC rate is separated by the error geometry observed on the output matrix. Figure 5b show the normalized DUE FiT for the GEMM on Kepler and Volta.

resources are saturated, the data in Figure 5 can be used to compare the GEMM reliability across architectures. Smaller matrices will lower the FIT rate due to the unused resources.

We also investigate the corrupted outputs of GEMM to classify the errors based on their coordinates (i.e., error geometry). Figure 5a shows the SDC rate separated into four types: (1) **Single** errors, only one corrupted element in the output matrix; (2) **Square**, four or more elements distributed in a rectangle; (3) **Line**, multiple corrupted elements on the same row/column of the output matrix; (4) **Random**, randomly distributed errors. In most 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. Instead, the impact of a single particle is *spread* during computation across multiple output elements. We use this information to simulate multiple faults when performing fault injections at the SASS level (see Section V).

From Figure 5b, it is clear that GEMM reliability depends on both the device and the data/operation precision. The DUE rate when ECC OFF is lower than ECC ON on Kepler. When there is a double-bit error, ECC triggers an application DUE. When ECC is OFF, double-bit flips may lead to an SDC at the output of GEMM.

We can notice that, independently of precision, the use of tensor cores increases DUE FIT. The GEMM with tensor cores is optimized to use the tensor cores to the maximum performance by performing Warp-level synchronizations. From the obtained results, the corruption of internal resources, necessary to perform specific Warp-level synchronizations of the tensor core, is particularly prone to generate a DUE. When the tensor cores are used to improve the performance of the DNNs, they may further increase the DNNs' DUE rate.

Figure 5a shows that GEMM with tensor cores has a lower SDC FIT rate for all SDC types when executed in FP16 precision for Volta GPU. This attests that the tensor core circuit is slightly more reliable than the combination of ADD, MUL, and the loop control variables needed to implement matrix multiplication in software. When GEMM is executed in FP32 precision, the tensor core SDC FIT rate increases significantly, being, on average for the SDC types, 20% higher than the software GEMM FIT rate. In fact, as stated in Section III, the tensor cores on Volta architecture execute physical operations only in FP16 precision. FP32 precision inputs require a hardware casting to FP16 precision, increasing the GPU's occupation and, thus, the SDCs (and DUEs) FIT rate. When executed on Volta without the tensor cores, the GEMM FIT rate increases as the precision increases. The increase from FP16 to FP32 is about $2\times$. As FP32 occupies $2\times$ more GPU resources, it is expected that the FIT rate will also have an increase with a factor close to $2\times$.

Square errors are potentially the most severe for DNNs. Square errors in GEMM are caused by faults that impact the scheduling or the execution of multiple threads in a Streaming Multiprocessor (SM). GEMM is a highly optimized version of matrix multiplication since it divides the input matrices into chunks (tiles) that fit in the cache of an SM, reducing memory latency. If a fault causes a thread to be incorrectly assigned or scheduled to an SM, or if some threads fail to synchronize, the whole SM output matrix part is likely to be corrupted, leading to a Square error. Errors in memory elements protected by ECC (e.g., registers and caches) manifest into either Single corrupted element or Line errors [57]. When ECC is turned ON, Single and Line errors (which are less critical for DNNs) are corrected, but the remaining errors (including rectangular errors) are not corrected. As a result, the percentage of Square errors increases when ECC is enabled.

Figure 5a also shows that the geometry of the output errors slightly changes when comparing Kepler and Volta. For FP32 GEMM on both GPUs with ECC ON, the error distribution is mainly Square and Single errors, showing that the criticality of the GEMM is directly related to how the error is propagated from the low-level fault to the algorithm output (details in Section V). It is worth noting that the criticality of the errors increases for the software GEMM compared to the GEMM optimized with the tensor core. The matrix multiplication performed in hardware generated more Single errors than the

version only performed in software. This result is in line with past results [14], where it has been shown that the reliability of DNNs hardware accelerators has a less critical error model than the operations performed only in software.

In the next section, we present the fault injection results using different fault models on GEMM and YOLOv3. We demonstrate that using only the naive single-bit flip at the SASS level fault injection is not enough to mimic the errors observed in beam experiments.

V. Modeling complex faults with fault injection

This section presents the fault injection results with the multiple fault models proposed in Section II. We show the Software Vulnerability Factor (SVF) for the injections performed on GEMM benchmarks for several configurations. To demonstrate the impact of the fault models on DNNs, we also show the YOLOv3 SVF for FP16 and FP32.

A. Fault impact on GEMM

Figure 6a shows the SVF for the fault injections performed on GEMM with Volta architecture, considering not only single-bit flips, but also more complex fault models. More precisely, we show results for four fault models, the standard Single Bit Flip, Single Random Value, Warp Zero Value, and Warp Random Value. The results are shown for two precisions, FP16 and FP32, with and without NVIDIA Volta tensor cores. The figure also presents the **Overall SVF**, obtained by combining all the fault models. Overall SVF serves as a reference to measure the impact of injecting a mix of faults instead of a single-bit fault model.

Results show that, despite the GEMM configuration, precision and algorithm, single-bit flip injections always produce single element corruption. However, as presented in Section IV, this model might not be realistic as an expected outcome from GEMM error criticality.

It is worth noticing that when a single random value is injected, it produces different outcomes on FP16 and FP32 GEMM. In fact, this happens because of how FP16 instructions are computed on the Volta architecture. Each FP32 GPU core on Volta can execute two FP16 operations simultaneously, and, after the computation, the result is stored in a 32-bit register. Consequently, when an output register is replaced by a random or a zero value on FP16, it will always generate at least two incorrect elements. Injecting in the most/least significant bit of the 16-bit for FP16 would produce single-element injections. However, previous work with FP16 microbenchmarks with neutron beam experiments [58] has shown that the fault model of FP16 instructions when executing on FP32 cores leads to double element corruption at the 32 bits registers. Thus, our decision to modify the entire output of FP16 instructions aims to produce more realistic results. Additionally, as random and zero values always produce multiple errors on FP16 injections, the warp-wide injections on FP16 will, consequently, produce a predominance of Square errors.

Another exciting outcome of the complex fault model injections is the differences between the GEMM performed only on software (multiply and accumulate instructions) and using tensor core instructions. For software-only GEMM, the outcomes of Warp-wide injections are 99% and 95% Square errors for FP16 and F32, respectively. However, when using tensor cores, 95% and 69% are Square errors for F16 and FP32, respectively. This is because when using tensor cores, the rounding mode is set to round towards zero [59], while the default operation is round to the nearest. The rounding method can mask the fault before it is propagated to the output. Similar behavior is observed on beam experiments when comparing only software GEMM and GEMM with tensor cores. The fault propagation of the GEMM can also be impacted by the order of the instructions and cast operations on FP32 when using tensor cores. For each configuration on Figure 6a, CUBLAS calls a very optimized kernel, which may lead to a different fault masking. Additionally, when performing FP32 Tensor cores GEMM on Volta architectures, a cast from FP32/FP16 before and after the Tensor Cores instruction are necessary, which can lead to a difference in the fault propagation when comparing the outcomes of FP32 and FP16 with Tensor Cores.

When we compare the Overall SVF for the FP16 and FP32 configurations, the FP32 produces more single errors than line and square, while the contrary happens for FP16. Similarly, the same configurations on beam experiments that use FP32 produce more single errors than the FP16 ones. In fact, the errors with FP16 are more critical than the ones with FP32, as smaller errors can be more easily masked when using FP32. It is worth noting that the high SVF produced on GEMM for complex fault models does not imply that they will generate only critical faults on a DNN. In the next section, we discuss the results of these fault models on YOLOv3.

B. Fault impact on YOLOv3

Figure 6b shows the SVF for the injections on YOLOv3 object detection DNN for two floating-point precisions, FP16 and FP32, on Volta architecture. We use the same fault models as for the GEMM fault injections, i.e., Single Bit Flip, Single Random Value, Warp Zero Value, and Warp Random Value. The figure also shows the Overall SVF for both precisions.

The differences between FP16 and FP32 are lower for YOLOv3 than for GEMM (FP32 Overall SVF is 25.7% while FP16 SVF is 27.6%). As expected, the reduced precision can change the error rate of YOLOv3 but impacts less the fault propagation (SVF). Contrarily, the GPU fault model significantly impacts the criticality of YOLOv3 errors. The most critical errors on YOLOv3 come from Single Random Value and Warp Random Value, 26.9% and 45.7% of critical SDCs with FP16, respectively, and 39.0% and 50.1% of critical SDCs with FP32, respectively. Similar behavior has been demonstrated with security attacks on DNNs, where the corruption of a single float exponent is enough to degrade the accuracy of some DNNs by 69% [60], [61]. Consequently, it is expected that the *Random Value* based fault models will produce more critical errors.

An exciting result from Figure 6b is the impact of the single-bit flip and Warp zero value fault models on YOLOv3. It is known from the GEMM fault injections that single-bit flip

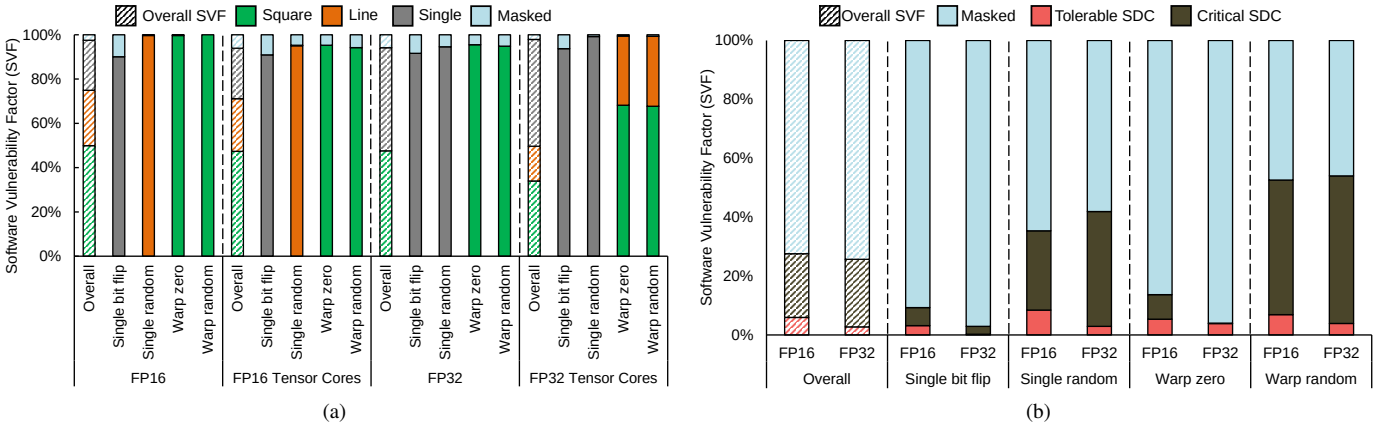


Fig. 6: Software Vulnerability Factor (SVF) for GEMM configurations on Figure 6a and YOLOv3 on Figure 6b. The outcome of each fault injection is grouped using the same methodology used for beam experiments results.

injections will generate single errors. Moreover, the single-bit flip on the lower places of the floating-point representation will generate small perturbations on YOLOv3 inference, leading to a low SVF for FP16 and FP32 precisions. The Single bit flip fault model produces an SVF of 9.3% for FP16 and 2.9% for FP32, and the Warp zero value produces an SVF of 13.7% for FP16 and 4.0% for FPF32. Correspondingly, zeroing DNN parameters can act more like a regularizer instead of reducing the accuracy of the network [62], [63]. The SVF for YOLOv3, when Warp Zero Value is injected, is 2.0× lower than the Overall SVF for FP16 and 6.4× lower than the Overall SVF for FP32.

Assuming that only one fault model is sufficient to produce a realistic DNN reliability evaluation is an oversimplification, since various types of faults are generated from the neutron iterations with the hardware in the beam experiments. To validate our findings, Figure 7 compares the critical SDCs rate measured with neutron beam experiments and with fault injection. As shown, when a cocktail of different fault models is injected, we observe a percentage of critical SDCs which is very similar to the beam experiments. The Critical SDCs of Overall SVF are 47% and 46% lower than the most critical fault model (Warp Random Value) for FP16 and FP32, respectively. The Critical SDCs of Overall SVF are 3.53× and 8.66× higher than Single bit flip for FP16 and FP32, respectively. These results confirm that injecting single-bit flip faults at the software level is unrealistic for the reliability evaluation of DNNs on GPUs, as it can underestimate the real SDC criticality.

Finally, we recall that our presented fault models are based on syndromes observed on RTL fault injections which we propagate to the SASS-level to explain the error criticality observed in neutron beam experiments. We are able to quickly simulate the sources of critical errors in software without having to employ fault injections using RTL or microarchitectural simulations. Lower level fault injections are in fact more accurate, but very slow. A simple GEMM evaluation for one configuration as presented in Figure 6a would take 4.8×10^5 hours to simulate according to recent works [34]. This would

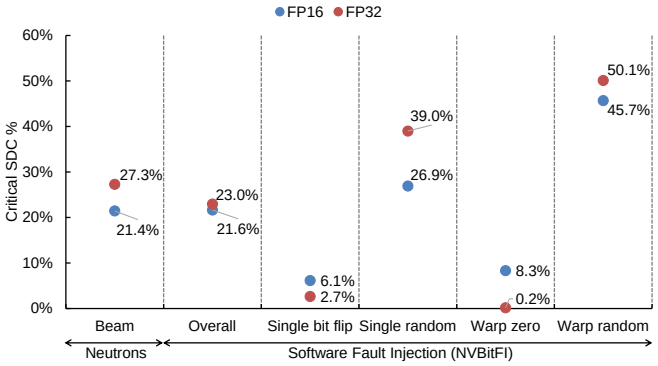


Fig. 7: Comparison between the YOLOv3 SDC criticality on neutron beam experiments vs. NVBitFI.

make the reliability evaluation of DNNs unfeasible. We acknowledge that our approach presents some limitations as the fault injection is based on SASS-level instrumentation, that is, software level. Consequently, our fault injection is limited to the user-accessible resources, making it impossible to simulate faults that affect resources like memory buffers. However, despite the proposed fault models' limitations, we can still simulate the syndrome of a fault in a shared resource of GPU faults at the SASS level.

VI. Conclusions

We have discussed the reliability of DNNs on beam experiments and evaluated the criticality of the errors in the output. Although the ECC is a powerful technique to reduce the SDC rate, it cannot reduce the Critical errors proportionally when DNNs are considered as an application. We then characterized the GEMM kernels as the core of DNNs, to investigate the error models that can impact the DNNs reliability. As radiation experiments demonstrated, the single bit flip in the memories is not the leading cause of critical errors. Additionally, the criticality of the output of the GEMM indicates that a single error in a single element of a GEMM kernel is not realistic as an error model to simulate critical errors on DNNs. We

then have performed a SASS-level fault injection on YOLOv3 object detection with different fault models to investigate the leading cause of the critical errors. We show that it is possible to simulate complex fault models at a higher level of fault injection, and that injecting single-bit flips at instruction output may not be the most realistic fault model for complex applications like DNNs.

Appendix A: Research Artifacts

In order to allow the reproducibility of the experiments performed in this paper, we have uploaded the source codes of our setups to online repositories on GitHub. The source code of our beam experiment setup is available at https://github.com/radhelper. Additionally, we have forked the NVBitFI repository at https://github.com/fernandoFernandeSantos/nvbitfi to include the FP16, Tensor Cores, and warp-wide fault models. A pull request was submitted on the official NVIDIA NVBitFI GitHub (https://github.com/NVlabs/nvbitfi/pull/19) to include the new fault models.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie (MSCA) grant agreement No 886202, from The Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001, from BIENVENÜE Bienvenue Fellowship MSCA Co-funded program, and ANR RETRUSTING (ANR-21-CE24-0015-02). Neutron beam time was provided by ChipIR (DOI:10.5286/ISIS.E.RB2200004-1, 10.5286/ISIS.E.RB2000137-1, 10.5286/ISIS.E.101136531) thanks to Chris Frost, Carlo Cazzaniga, and Maria Kastriotou and by LANSCE thanks to Steve Wender and Gus Sinnis.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *14th European Conference on Computer Vision – ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 630–645.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 91—-99.

[3] G. Gkioxari, J. Johnson, and J. Malik, "Mesh R-CNN," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 9784–9794.

[4] NVIDIA. (2017) NVIDIA Tesla V100 GPU architecture. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[5] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 6–18.

[6] J. C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," in *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years.*, Jun 1995, pp. 10–22.

[7] M. Nicolaidis, "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies," in *17th IEEE VLSI Test Symposium (VTS '99), 25-30 April 1999, San Diego, CA, USA*. IEEE Computer Society, 1999, pp. 86–94.

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

[9] S. Mukherjee, J. Emer, and S. Reinhardt, "The soft error problem: an architectural perspective," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, February 2005, pp. 243–247.

[10] 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.

[11] 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 96–108.

[12] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in DNN accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, pp. 969–988, 2020.

[13] S. Blower, P. Rech, C. Cazzaniga, M. Kastriotou, and C. D. Frost, "Evaluating and Mitigating Neutrons Effects on COTS EdgeAI Accelerators," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1719–1726, 2021.

[14] R. L. Rech Junior, S. Malde, C. Cazzaniga, M. Kastriotou, M. Letiche, C. Frost, and P. Rech, "High Energy and Thermal Neutron Sensitivity of Google Tensor Processing Units," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 567–575, 2022.

[15] N. DeBardeleben, S. Blanchard, L. Monroe, P. Romero, D. Grunau, C. Idler, and C. Wright, "GPU Behavior on a Large HPC Cluster," in *Euro-Par 2013: Parallel Processing Workshops*, D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 680–689.

[16] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342.

[17] M. B. Sullivan, N. Saxena, M. O'Connor, D. Lee, P. Racunas, S. Hukerikar, T. Tsai, S. K. S. Hari, and S. W. Keckler, *Characterizing And Mitigating Soft Errors in GPU DRAM*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 641—-653.

[18] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 304–311.

[19] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "NVBitFI: Dynamic Fault Injection for GPUs," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[20] N. N. Mahatme, S. Jagannathan, T. D. Loveless, L. W. Massengill, B. L. Bhuva, S.-J. Wen, and R. Wong, "Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process," *IEEE Transactions on Nuclear Science*, vol. 58, no. 6, pp. 2719–2725, 2011.

[21] F. F. d. Santos, J. E. R. Condia, L. Carro, M. S. Reorda, and P. Rech, "Revealing GPUs Vulnerabilities by Combining Register-Transfer and Software-Level Fault Injection," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 292–304.

[22] L. K. Draghetti, F. F. d. Santos, L. Carro, and P. Rech, "Detecting Errors in Convolutional Neural Networks Using Inter Frame Spatio-Temporal Correlation," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019, pp. 310–315.

[23] S. Jha, S. Cui, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Exploiting Temporal Data Diversity for Detecting Safety-critical Faults in AV Compute Systems," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 88–100.

[24] Y. Long, X. She, and S. Mukhopadhyay, "Design of Reliable DNN Accelerator with Un-reliable ReRAM," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1769–1774.

[25] S. Mittal, "A survey on modeling and improving reliability of DNN algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, pp. 689–716, 2020.

[26] G. Li, K. Pattabiraman, C. Y. Cher, and P. Bose, "Understanding Error Propagation in GPGPU Applications," in *SC16: Int. Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016, pp. 240–251.

[27] D. A. G. Gonçalves de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and Mitigation of Radiation-Induced Soft Errors in Graphics Processing Units," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 791–804, 2016.

[28] G. León, J. M. Badía, J. A. Belloch, A. Lindoso, and L. Entrena, "Evaluating the soft error sensitivity of a GPU-based SoC for matrix multiplication," *Microelectronics Reliability*, vol. 114, pp. 856 – 861, 2020, 31st European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, ESREF 2020.

[29] 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.

[30] J. M. Badia, G. Leon, J. A. Belloch, M. Garcia-Valderas, A. Lindoso, and L. Entrena, "Comparison of Parallel Implementation Strategies in GPU-Accelerated System-on-Chip Under Proton Irradiation," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 444–452, 2022.

[31] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How Reduced Data Precision and Degree of Parallelism Impact the Reliability of Convolutional Neural Networks on FPGAs," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 865–872, 2021.

[32] X. Iturbe, B. Venu, and E. Ozer, "Soft error vulnerability assessment of the real-time safety-related ARM Cortex-R5 CPU," in *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2016, pp. 91–96.

[33] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 711–721.

[34] 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, pp. 660–674, 2020.

[35] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, and S. Di Carlo, "RT Level vs. Microarchitecture-Level Reliability Assessment: Case Study on ARM(R) Cortex(R)-A9 CPU," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2017, pp. 117–120.

[36] C. Constantinescu, M. Butler, and C. Weller, "Error injection-based study of soft error propagation in AMD Bulldozer microprocessor module," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012, pp. 101–106.

[37] D. Ferraretto and G. Pravadelli, "Simulation-based Fault Injection with QEMU for Speeding-up Dependability Analysis of Embedded Software," *Journal of Electronic Testing*, vol. 32, no. 1, pp. 43–57, 2016.

[38] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 221–230.

[39] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: A Runtime Perturbation Tool for DNNs," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 25–31.

[40] L. Yang, B. Nie, A. Jog, and E. Smirni, "Enabling Software Resilience in GPGPU Applications via Partial Thread Protection," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 1248–1259.

[41] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and Accurate Error Simulation for CNNs Against Soft Errors," *IEEE Transactions on Computers*, vol. Early Access, p. Early Access, June 2022.

[42] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003, pp. 29–40.

[43] G. Papadimitriou and D. Gizopoulos, "Demystifying the System Vulnerability Stack: Transient Fault Effects Across the Layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 902–915.

[44] K. Andryc, M. Merchant, and R. Tessier, "FlexGrip: A soft GPGPU for FPGAs," in *2013 International Conference on Field-Programmable Technology (FPT)*, 2013, pp. 230–237.

[45] 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 *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 249–258.

[46] C. Lunardi, F. Previlon, D. Kaeli, and P. Rech, "On the Efficacy of ECC and the Benefits of FinFET Transistor Layout for GPU Reliability," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1843–1850, 2018.

[47] NVIDIA. (2012) Kepler GK110/210 Whitepaper. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

[48] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[49] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[50] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678.

[51] R. Collobert, K. Kavukcuoglu, and C. Farabet. (2011) Torch7: A Matlab-like Environment for Machine Learning. [Online]. Available: http://infoscience.epfl.ch/record/192376

[52] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[53] C. Cazzaniga and C. D. Frost, "Progress of the Scientific Commissioning of a fast neutron beamline for Chip Irradiation," *Journal of Physics: Conference Series*, vol. 1021, pp. 159–163, may 2018.

[54] C. Slayman, *JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors*. Boston, MA: Springer US, 2011, pp. 55–76.

[55] D. Oliveira, L. Pilla, N. DeBardeleben, S. Blanchard, H. Quinn, I. Koren, P. Navaux, and P. Rech, "Experimental and Analytical Study of Xeon Phi Reliability," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 336–348.

[56] S. Jha, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. Kalbarczyk, S. W. Keckler, and R. K. Iyer, " Kayotee: A Fault Injection-based System to Assess the Safety and Reliability of Autonomous Vehicles to Faults and Errors," *CoRR*, vol. abs/1907.01024, 2019. [Online]. Available: http://arxiv.org/abs/1907.01024

[57] P. Rech, T. Fairbanks, H. Quinn, and L. Carro, "Threads Distribution Effects on Graphics Processing Units Neutron Sensitivity," *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4220–4225, Dec 2013.

[58] F. Fernandes dos Santos, C. Lunardi, D. Oliveira, F. Libano, and P. Rech, "Reliability Evaluation of Mixed-Precision Architectures," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 238–249.

[59] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA Tensor Core Programmability, Performance & Precision," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 522–531.

[60] A. S. Rakin, Z. He, and D. Fan, "Bit-Flip Attack: Crushing Neural Network With Progressive Bit Search," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220.

[61] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, "Defending Bit-Flip Attack through DNN Weight Reconstruction," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1022–1028.

[62] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[63] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2217–2225.