HAL open science

Reliability Assessment of Large DNN Models: Trading Off Performance and Accuracy

Junchao Chen, Giuseppe Esposito, Fernando Fernandes dos Santos, Juan-David Guerrero-Balaguera, Angeliki Kritikakou, Milos Krstic, Robert Limas, Josie E Rodriguez Condia, Matteo Sonza Reorda, Marcello Traiola, et al.

▶ **To cite this version:**

Junchao Chen, Giuseppe Esposito, Fernando Fernandes dos Santos, Juan-David Guerrero-Balaguera, Angeliki Kritikakou, et al.. Reliability Assessment of Large DNN Models: Trading Off Performance and Accuracy. VLSI-SoC 2024 - IFIP/IEEE International Conference on Very Large Scale Integration, Delft University of Technology and University of Abdelmalek Saadi University, Oct 2024, Tanger, Morocco. pp.1-10, 10.5286/ISIS.E.RB2300036) . hal-04736733

HAL Id: hal-04736733

https://hal.science/hal-04736733v1

Submitted on 15 Oct 2024

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.

Distributed under a Creative Commons Attribution 4.0 International License

Reliability Assessment of Large DNN Models: Trading Off Performance and Accuracy

Junchao Chen[1], Giuseppe Esposito[2], Fernando Fernandes dos Santos[3], Juan-David Guerrero-Balaguera[2], Angeliki Kritikakou[3], Milos Krstic[1,4], Robert Limas Sierra[2], Josie E. Rodriguez Condia[2], Matteo Sonza Reorda[2], Marcello Traiola[3], and Alessandro Veronesi[1]

[1]IHP – Leibniz Institute for High-Performance Microelectronics, Germany
[2]Politecnico di Torino, Department of Control and Computer Engineering (DAUIN), Turin, Italy
[3]Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France
[4]University of Potsdam, Germany

***Abstract*—The adoption of Deep Neural Networks (DNNs) in several domains allows for increased effectiveness in applications that deal with massive data-intensive and complex data inputs. When employed in safety-critical scenarios, such as automotive, aerospace, healthcare, and autonomous robotics, assessing the DNNs' reliability and functional safety is crucial to ensure their correct in-field operation, even in the presence of hardware faults. However, the system complexity and the massive amounts of data to be processed by DNNs prevent the effective adoption of traditional strategies for reliability characterization and for identifying the most fault-sensitive structures. Accurate fault assessment strategies usually require unacceptable computational power and large evaluation times. On the other hand, faster strategies commonly lack accuracy in correctly representing system faults. Consequently, it is necessary to develop effective strategies that trade-off between performance and accuracy.**

This work analyses three reliability assessment strategies for deep neural networks and their underlying hardware, highlighting the main solutions and challenges in terms of evaluation performance and fault characterization accuracy. We overview different solutions to evaluate the hardware accelerators implementing DNNs at three abstraction levels: *i)* by physically injecting faults on a GPU running DNNs, *ii)* by performing microarchitectural characterization of GPUs to develop application-accurate error models, and *iii)* by using structure-aware cross-layer error modeling on DNN hardware accelerators. Our experimental results indicate that accurate error representation requires structural features from the targeted hardware.

I. Introduction

Deep Neural Networks (DNNs) are widely used in various fields to enhance the performance of complex applications, such as autonomous driving, natural language processing, and industrial robots. These applications typically involve processing complex operations and large volumes of data (i.e., $1x10^6$ operations per second) [1], [2]. When employed in safety-critical applications, such as aerospace, automotive, and healthcare, DNN-based solutions must match strict reliability requirements mandated by industrial standards (i.e., ISO 26262 and ISO/IEC 22989 [3], [4]). In safety-critical applications, reliability characterization and evaluation are mandatory to ensure that any fault/defect arising in the system can be controlled and properly handled. These reliability assessments may guide designers during early design stages and contribute to achieving the target fault tolerance.

DNN accelerators, including *Graphics Processing Units* (GPUs), are built with advanced technology node processes that enable the design of smaller, faster, and energy-efficient devices [5]. Unfortunately, miniaturization (7nm or below) seriously increases the reliability concerns in a system connected to temporal (aging and wear-out) and environmental variations (over-stress, environmental harshness due to temperature, voltage, or radiation) [6]–[8]. These variations increase the fault rate, impacting the hardware and propagating the faults to the application as *Silent Data Corruptions* (SDCs), jeopardizing the in-field operation of the complete system [9]–[11].

To assess the impact of faults in a DNN-based system, several reliability evaluation strategies have been developed, such as formal techniques, simulation-based architectural evaluations, and physical/beam experiments. Formal evaluation and software simulation strategies focus their analyses on the DNN model with feasible evaluation times but neglect the underlying hardware. In contrast, other strategies inject faults at the hardware level and can provide fine-grain accuracy, i.e., at the gate or micro-architectural level. However, hardware simulations for large DNNs often lead to years/decades of expected simulation time. Hence, DNN's complexity, operational density, and underlying hardware demand the development of effective strategies to provide efficient evaluation times under feasible levels of fault characterization accuracy [12]. Inspired by the performance bottlenecks in some evaluation strategies and the challenges in correctly characterizing fault impacts on DNN-based systems, *we analyze and discuss the main advantages and challenges of three strategies to*

This work was partially supported by the Italian National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing, and by the Federal Ministry of Education and Research of Germany under the programme of "Souverän. Digital. Vernetzt." Joint project 6G-RIC, project identification number: 16KISK026.
This activity received funding from the European Union's 2020 research and innovation programme under grant agreement No 101008126 (RADNEXT project), and by the ANR FASY (ANR-21-CE25-0008-01). ChipIR provided and supported neutron beam time experiments (DOI https://doi.org/10.5286/ISIS.E.RB2300036).

assess the reliability of DNNs. These strategies consider the structural hardware characteristics while keeping a good trade-off between evaluation time and fault simulation details.

The first strategy (**Section III**) combines physical characterization using a beam of neutrons and software-based fault simulation on real devices to evaluate the impact of transient faults on large DNN models for image classification. Additionally, we show that an evaluation that considers the hardware characteristics is mandatory to deploy efficient software fault tolerance for large DNNs. A second strategy (**Section IV**) combines the structural features of a GPU and its units (i.e., schedulers and *'TCUs'*) to characterize fault effects at the system level. This strategy determines representative hardware-aware fine-grain corruptions to identify accurate error models for evaluating large DNNs at the software level. Finally, the third strategy (**Section V**) develops a cycle-accurate micro-architecture model of the NVDLA architecture, incorporating fine-grained structural details to ensure accurate fault characterization for large DNN workloads, and shows that variations in hardware parameters significantly impact both the types and occurrence of observed errors, depending on the specific DNN model.

We explore the benefits of combining two or more layers of fault injection abstraction to effectively assess and harden large and complex DNNs within acceptable evaluation times. The strategies we discuss are intended to support quick system improvements and efficient hardening.

II. Challenges in DNN reliability assessment

When GPUs are used as DNN accelerators in autonomous systems, they are susceptible to multiple sources of failures, such as ionizing radiation, permanent faults, performance degrading faults, timing errors, and intermittent faults [13], [14]. These sources of faults may not make the GPU completely inoperative, but they can significantly impact the execution of DNN models, potentially changing the final inference result. When not **masked**, faults can become errors that can propagate to the software level and lead to failures such as the **Detected Unrecoverable Errors (DUEs)**, which hang the program or crash the entire system, and **Silent Data Corruptions (SDCs)**, that allow the application to complete its execution but with an incorrect output. *Without a proper reliability assessment guiding the introduction of proper fault-tolerance solutions, the resulting failure remains undetected*. When considering DNN models, SDCs can be further categorized into **Tolerable SDCs**, which modify the model output but not the inference outcome (i.e., the classification, detection, or segmentation), or **Critical SDCs**, which cause the model to change the inference probabilities, resulting in mis-classification, mis-detection, or mis-segmentation.

In order to characterize the hardware faults that reach the highest levels of a system and become failures, different approaches can be used, each one having its own advantages and disadvantages. We can separate the reliability assessment methods into three categories: *physical fault injection*, *fault simulation techniques*, and *software-based fault simulation*.

Physical fault injection require special facilities to expose the system to a beam of particles and induce faults in hardware. This strategy is an efficient and realistic evaluation approach for safety/mission-critical systems that operate in harsh environments, such as space and avionics applications. As the faults are physically injected into the circuit, physical fault injection effectively characterizes radiation-induced transient faults and permanent faults from accumulated radiation dose [15]–[17]. However, the strategy can hardly provide information regarding specific sub-structures in a system or a DNN model. For instance, radiation experiments do not allow fault propagation analysis since failures are only observed at the output, making it hard to identify the error sources in hardware or software parts in a system, as well as hindering the error modeling [18]. These constraints impede the straightforward identification of the most vulnerable parts of the system.

Hardware fault simulation techniques can be performed at several abstraction levels, from low-level micro-architecture (i.e., Register-Transfer or gate levels) [19] to architecture/functional evaluations [20], [21]. These strategies measure the fault propagation probability on the system, i.e., the *Architectural Vulnerability Factor* [22]. Hardware fault simulations have a fine-grain accuracy and can support the identification of the most fault-vulnerable hardware structures. However, for the hardware fault simulations, the computational power and evaluation times are proportional to the fine-grain abstraction, the system's size, and complexity. Thus, the complexity and time required for a detailed evaluation can require an unacceptable time, e.g., $> 10,000$ days for a small DNN evaluation [14].

Software-based fault simulation consists of code instrumentation or/and modification strategies that add instructions or routines to a targeted application to allow faults injection in the underlying hardware architecture [23], [24]. Faults injected in software fault simulations can be used to estimate the Program Vulnerability Factor (PVF), i.e., the probability for a fault to propagate [25]. Software-based strategies usually use real execution time and can be performed on actual hardware platforms, leading to fast evaluations. The main challenge of software fault simulation strategies is the accuracy of identifying and describing faults from the underlying hardware. The user defines the fault model, which risks not being realistic, leading to inaccurate results. Additionally, the evaluated faults can only be injected into a subset of available and accessible resources at the software level (e.g., registers or variables).

The following sections demonstrate how to effectively and efficiently evaluate large DNN models by combining two or more reliability assessment approaches without compromising accuracy or incurring prohibitively long evaluation times.

III. DNN's Reliability Evaluation Through Physical and Software Fault Injection

In this section, we show how combining two experimental evaluation methods – physical fault injection using a neutron beam and software fault simulation – enables the efficient evaluation and hardening of large DNN models without compromising accuracy. We start by summarizing the experimental

methodologies, and then we discuss the general trends observed in the experiments and their importance.

A. Experimental Methodologies

We performed fault injection using a software fault simulator and a neutron beamline. For all the experiments, we evaluated 5 ViT models without any protection, as well as the same models protected by a range restriction strategy. In the employed range restriction, the ViT parameters are checked to see if they fall within a range of accepted values when propagating through the ViT's Identity layers. If they do, they are propagated as usual. If not, they are replaced by 0. We evaluate both versions of the ViT models to demonstrate why it is mandatory to consider both hardware and software analysis when proposing a fault tolerance method for DNNs.

*1) **Software fault simulation**:* We used the NVIDIA Bit Fault Injector (NVBitFI) [24] for instruction-level fault simulation. NVBitFI enables the simulation of faults at the Shader Assembly level (SASS), which means at the assembly level of GPU kernels. With NVBitFI, we could select different fault models and sites to evaluate the ViT models.

We injected faults in general-purpose registers, memory load instructions, and arithmetic floating-point operations. We simulated 1,750 faults per ViT model. The failures (SDCs and DUEs) were counted similarly to beam experiments. Through fault simulations, we calculated the Program Vulnerability Factor (PVF) for each ViT model. The PVF represents the probability of an injected fault propagating from the assembly instruction to the application output [25].

We performed fault simulations with single-bit flip, random value, and warp random value fault models. The first two fault models change an instruction's output register by flipping a bit or replacing it with a random value. While injecting single-bit flips at software is a widely used fault model, it has been demonstrated to incorrectly model fault impact for complex applications [26]. Injecting an experimentally tuned fault model in software (i.e., the observed manifestation of the hardware fault in a software visible state) has been proven to be accurate for GPUs [12], [27]. Consequently, faults are also injected at the warp level using NVBitFI. The warp level fault model is based on recently proposed fault models for GPUs [19], [27], where the outputs of the same instruction in all the threads within a warp are corrupted.

*2) **Beam experiments**:* Experimental tests were conducted at ChipIr in Rutherford Appleton Laboratory (RAL, UK). The facility provides a neutron beam to replicate atmospheric neutron effects on electronic devices [28]. This allows for the realistic measurement of device failure rates while running a code. Figure 1 shows the installed setup, which consists of GPUs aligned with the neutron beam and connected to the motherboard. Python scripts run on a server outside the beam monitor and launch the ViT models on the devices inside the beam room. The software is designed to recover from device hangs and restart the program if it fails to respond within a specified timeframe. The same ViT model is run on the GPU for several iterations, and any differences between the output

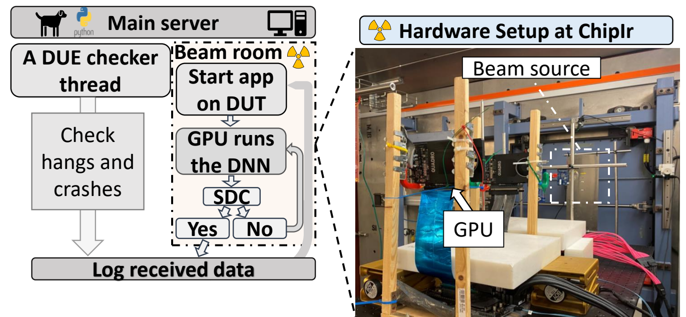


Fig. 1: Software and hardware setups for the neutron beam experiments at ChipIr. The server located outside the beam room controls the devices exposed to the neutron beam. Scripts monitor any disturbances (SDCs and DUEs) while the ViT models perform inferences on the GPUs.

TABLE I: ViT models size, accuracy on Imagenet dataset, and execution times for Pascal GPU.

	Config.	Size (MB)	Accuracy (%)	Time (ms)
ViT-H [29]	H14-224	2479	88.20	1644
EVA2 [30]	L14-448	1176	89.95	2686
SwinV2 [31]	L-256	787	86.94	404
MaxViT [32]	L-384	845	87.98	938

and a previously saved output (fault-free golden) are recorded as Tolerable SDC or Critical SDC. We make all the codes used in the beam experiments available [1].

The beam experiments measured the probability of a neutron causing a failure in the GPU. The failure rate determined in the experiments can be used to estimate the terrestrial failure rate caused by neutrons on a GPU. The beam experiments provide the Failure In Time (FIT) - the number of faults expected in 10^9h of operation. FIT is calculated by dividing the number of errors by the neutron fluence, then multiplying by the terrestrial neutron flux $(13n/(cm^2 \times h))$ and by 10^9.

*3) **Device and DNNs Under Test**:* For the beam experiments, the NVIDIA GPU Pascal architecture (Quadro P2000) was used. The Quadro P2000 is built with TSMC $16nm$ FinFET, featuring an L1 cache of 48KB per Streaming Multiprocessor (SM), an L2 cache of 1280 KB, and 1024 CUDA cores. The GPU has 256 KB registers per SM and a power consumption of up to 75W. Our beam experiments only focus on GPU core errors (beam spot set to 2cm diameter to avoid affecting onboard DRAM).

We evaluated 5 ViT models from the HuggingFace library [33]. The models belong to 4 families: Original ViT [29], EVA2 [30], SwinV2 [31], and MaxViT [32]. The models differ in size and input patches. For the experiments, we used a Python program with PyTorch to load the ViT and perform inferences on a batch of random images from the ImageNet dataset [34].

B. Software Fault Simulation vs Beam Experiments

Fault simulation and lower levels of fault injection (such as physical or microarchitectural) can yield significantly different

[1] https://github.com/diehardnet/maximals

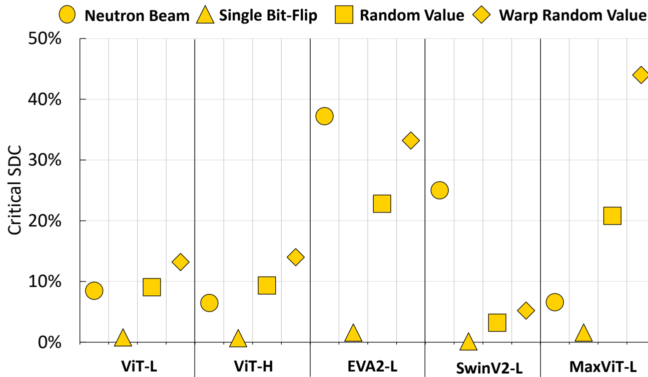


Fig. 2: Comparison of Critical SDC percentages between software fault simulations and neutron beam experiments. We used various fault models for broad analysis.

results [26]. Recent studies have shown that this difference can be orders of magnitude [26], [35]. To measure the differences between the beam experiments and the software fault injection for ViTs, we start our analysis by evaluating the impact of the faults on the Critical SDCs.

Figure 2 shows the percentages of Critical SDCs for both NVBitFI and ChipIr neutron beam experiments of all the ViTs listed in Table I. In all cases, the single-bit flip underestimated the percentages of Critical SDCs compared to the beam experiments. On average, 0.80% of the injections with a single bit flip generated a Critical SDC, while 13.95% of the observed SDCs in the beam experiments were critical. Similar to other types of DNNs, ViTs are resistant to single-bit flips caused by transient faults [36]. The effect of a single bit in a single parameter on models containing millions of parameters is expected to be minimal. More complex fault models resulted in a significantly higher percentage of Critical SDCs, with 10.87% for random value and 18.27% for warp random value. Our data indicates that more complex fault models are mandatory to evaluate large ViTs' reliability accurately.

Single-bit flips injected at the instruction level do not provide a realistic ViT evaluation since most are masked and produce a Critical SDC rate close to zero, well off the rate obtained with beam experiments. We observed on the fault simulation that only 3% of single-bit flip fault injections resulted in values higher than 10^6 after the fault mask was applied to the target register. In contrast, when random values were used, 45% of the fault mask immediately produced values higher than 10^6, NaN, or infinity values. If these faults spread to ViT structures, they may lead to Critical SDCs. To protect the ViTs against those faults, we employed float value restriction on the Identity layers of the selected models. Value restriction limits the maximum and minimum values the DNN parameters can propagate, filtering the values corrupted by faults. Value restriction is a standard method to improve the fault tolerance of DNNs with a low overhead [37].

Table II presents the Critical SDC comparison between the Baseline and Hardened ViT models, assessed using NVBitFI (Critical SDC PVF) and the ChipIr (Critical SDC FIT rate)

TABLE II: Comparison of the results obtained from software fault simulation and beam experiments for both the unhardened models (Baseline) and the models protected by range restriction (Hardened).

		NVBitFI [PVF]	ChipIr [FIT]	Reduction NVBitFI	Reduction ChipIr
ViT-L	**Baseline**	6.29%	1.77±0.45	3.67×	1.69×
	Hardened	1.71%	1.05±0.26		
ViT-H	**Baseline**	6.11%	1.61±0.33	2.74×	3.20×
	Hardened	2.23%	0.50±0.13		
EVA2-L	**Baseline**	15.20%	3.95±1.18	10.64×	–
	Hardened	1.43%	0.00±1.99		
SwinV2-L	**Baseline**	2.17%	0.87±0.60	1.23×	1.74×
	Hardened	1.77%	0.50±0.21		
MaxViT-L	**Baseline**	15.89%	1.75±0.36	1.94×	4.61×
	Hardened	8.17%	0.38±0.07		

neutron beam. The Critical SDC reduction (Baseline/Hardened) is presented for both NVBitFI and ChipIr experiments. During the beam experiment campaign, no Critical SDC was observed for EVA2-L.

By employing a complex fault model in the software simulation, we are able to design a fault tolerance that is capable of preventing Critical SDCs on the beam experiments. Table II shows that the hardened versions of the ViT models produced, on average, 2.81× less Critical SDCs compared to the unhardened version on the ChipIr beam experiments. Similarly, the NVBitFI, using a set of complex fault models (including single-bit flip, random values, and warp random values), shows a 2.40× reduction on the Critical SDC.

Physical fault injection with a beam of particles is a method widespread in industries such as space exploration and avionics, and they are also used as a validation method for academic research. However, the high cost (thousands of dollars per hour), complexity, and limited availability (facilities have scheduling windows of months) hinder a more detailed evaluation of complex systems such as DNN-based systems. As a result, fault simulation is employed to model faults and errors at various hardware and software abstraction levels. Using fault simulation, researchers and engineers are not limited to transient or permanent faults caused by ionizing particles but also transient and permanent faults from other sources, such as aging, voltage variations, and performance degrading faults. The following sections will explore different methods for simulating faults on DNNs.

IV. Modeling Errors from Faulty TCUs in GPUs

A. *TCUs in GPUs*

Modern GPUs are equipped with specialized in-chip accelerators (e.g., Tensor Core Units, or TCUs) to boost the computation of *General Matrix Multiplication* (GEMM) algorithms, representing the fundamental building blocks for the efficient implementation of DNNs, including cutting-edge models, such as transformers or Large Language Models [38], [39]. Moreover, optimized libraries (such as cuBLAS for NVIDIA GPUs) use the GEMM algorithm to reshape the kernel weights and feature maps as matrices [40] for their later deployment in GPUs. In particular, these libraries split

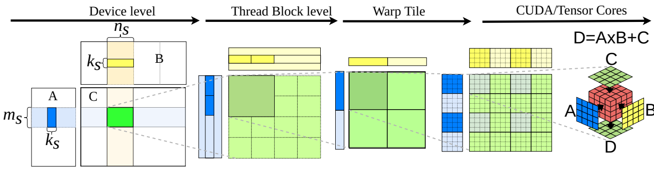


Fig. 3: A scheme of the tiling-based GEMM execution in TCUs.

matrices into *tiles* and exploit the TCUs inside a GPU, splitting tiles at the device level into sub-tiles at the thread block, warp, and TCU levels, as depicted in Fig. 3 for the matrix operation A×B+C [41]. In detail, each *tile* is assigned to a unique cooperative thread group (CTA), which accelerates the computations by distributing all CTAs among the available parallel cores (SMs) in a GPU.

The widespread use of TCU-based GPUs for GEMM acceleration raises reliability concerns due to the large size of DNNs, and the increasing complexity of the underlying hardware [14], [42]. This section shows an efficient and accurate strategy to characterize the corruption effects from faults in the TCUs in terms of software-level error models, thus supporting application-level evaluations.

B. A method to model errors produced by faults in TCUs

Our strategy for software-level error modeling consists of determining *bit-flip-masks* that represent the corruption effects of the faults affecting the TCUs on the outputs of GEMM operations. Our strategy includes four steps: *i)* structure-aware fault characterization, *ii)* fault analysis and error modeling, *iii)* error generation, refinement and compression, and *iv)* model enhancement and validation (Fig. 4).

*1) **Structure-aware fault characterization**:* a seed GEMM operation describes a normal distribution of synthetic data with statistical information on the weights and the intermediate features from a target DNN model. The GEMM operation is deployed and executed on an instruction-accurate architectural simulator of TCUs in GPUs, which includes fault injection (FI) infrastructure for reliability evaluation [38], [39], [43]. Then, FI campaigns are conducted on the TCU's data-path structures.

*2) **Error generation**:* The fault effects are analyzed to determine **spatial** and **scalar** effects on the GEMM's outputs. The spatial evaluation finds the likelihood of every GEMM's output element (*location*) when corrupted by a faulty TCU. Furthermore, the scalar evaluation measures the bit-flip rate of the corrupted outputs on the GEMM computation. In particular, we consider two factors that corrupt a GEMM's output: *i)* the matrix tile's size, and *ii)* the ability of the data to activate and propagate faults. Thus, propagation effects are observed at the GEMM output as data corruptions in some locations of the executed tiles (i.e., from 1 to 8 corrupted elements per GEMM's tile at the warp level [38], [39]). Other factors, such as the number of SMs and the scheduling policy, directly depend on the targeted GPU (i.e., a faulty TCU inside an SM might induce data corruption only into those CTAs/tiles executed in the affected SM) [43].

The spatial effects are computed as *corruption probabilities* through a bi-dimensional representation in which every tile

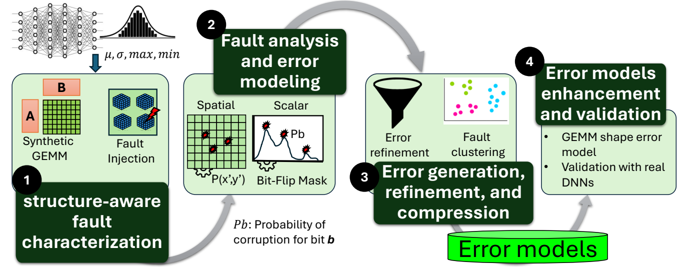


Fig. 4: A general scheme of the proposed strategy.

location reports the probability of a fault to induce corruption. In detail, we employ a common coordinate system (x',y'), an indicator function $\mathbb{1}_{(x',y')}(x,y)$ [44], and the experimental results to indicate spatially the corrupted GEMM's locations. Then, the probability per fault $P(x',y')$ is calculated as the ratio between the number of observed effects in the (x',y') element and the total number of executed tiles in a faulty SM.

At the scalar level, we evaluate the impact of permanent faults in TCUs as the changes on one or multiple bits of the final value of the computed GEMM operation by resorting to *bit-flip probabilities*. These correspond to the probability of corruption per bit (P_b), i.e., to the number of times the bit b changes w.r.t. the golden values divided by the total number of corrupted bits. It must be noted that the *bit-flip probabilities* can be adapted to specific number formats (e.g., floating-point or integer). Both probabilities (*corruption* and *bit-flip*) are combined to generate corruption masks (*bit-flip-masks*) to describe errors from hardware faults directly on the outputs of the GEMM operation.

In this work, we focused on floating-point formats, so bits are grouped as *Sign*, *Exponent* and *Mantissa*, which allow the identification of bit probabilities per group (e.g., *Probability of Exponent Corruption* or PEC, which is the ratio between faulty values with at least a one-bit flip in the exponent over the total number of elements affected by the fault) and simplify the definition of injection masks correctly describing identical effects from permanent faults in TCUs. From experimental results, we calculate metrics to identify corruption masks in the exponent and mantissa, such as PEC. Then, we combine error generation and refinement steps by first processing any propagated fault as a GEMM error. Afterward, we evaluate the error quality in a refinement routine. A complementary error compression step optimizes the number of errors for application-level evaluations. In our strategy, each fault causing a spatial GEMM corruption in the generic *element* x, y is evaluated through an iterative process. The strategy classifies the corruption probability per affected location ($P(x',y') \neq 0$) as a feasible target for error generation. Then, a *bit-flip-mask* is generated for the given error and associated with the allocated CTAs/tiles (*GEMM coordinate*) on a faulty TCU.

It must be noted that one or more *bit-flip-masks* can be associated with each fault. Moreover, some error models might produce equivalent spatial corruptions (i.e., sharing an identical spatial distribution but corrupting a value differently). In both cases, refinement and compression processes compact

the errors while preserving the quality and accuracy for the represented scalar error corruptions as *bit-flip-masks*.

*3) **Error refinement**:* in this step we use two complementary and focused evaluations using new randomly generated GEMM seeds. The first evaluation employs FI campaigns targeting all faults that caused error effects, while the second one uses the initial *bit-flip-masks* to place corruptions on GEMM operations. Then, we compute the *Mean Absolute Error* or (MAE) for corruptions from the FI campaigns (MAE_{faulty}) and from the error models (MAE_{masks}), evaluating the error replication accuracy by comparing (MAE_{faulty} and MAE_{masks}). Errors over- or under-estimating the scalar corruption effects (i.e., larger or lower MAE's magnitude than standard) are discarded, and their faults are considered *"unmodelable"*. Then, a refinement *Threshold* (*Th*, maximum discrepancy between corruptions from the proposed strategy w.r.t. the standard evaluation) is applied as ($1/Th < MAEs_{mask}/MAEs_{fault} < Th$) and experimentally tuned to trade-off corruption accuracy and number of represented errors from hardware faults.

*4) **Error Compression**:* this step compacts similar error models into clusters by considering their GEMM location and corruption magnitude. First, we group error models with the same corrupted GEMM region. Then, we compare the MAE per *bit-flip-masks* with the *(Th)* to organize them as clusters. A low value of *Th* provides higher error corruption accuracy (more clusters) but a lower compression, while a large value of *Th* allows higher compression (fewer clusters) at the expense of some loss in the error modeling accuracy. This compression also optimizes the overall error evaluation time since its main idea is to employ only one error per cluster instead of several errors with equivalent effects during application-level evaluations.

*5) **Model enhancement**:* this step focuses on improving and validating the effectiveness of the error models for different GEMM scenarios (i.e., sizes) against the conventional FI campaigns. Since the GEMM size, the number of tiles (T), and the reuse of faulty TCUs impact the distribution and accuracy of erroneous outputs, we resort to shape-wise error models to enhance the accuracy of errors according to the GEMM's shape. For this purpose, we compress error models for several GEMM shapes (e.g., $200 \times 200 \times 200$) using GEMM seeds. Then, we validate the accuracy of the shape-wise error models using convolutional layers from typical DNNs.

The validation involves two evaluations per layer: *1)* conventional FI campaigns in the TCUs and *2)* error evaluations using the shape-wise error models, injecting only one *bit-flip-mask* per cluster. The MAEs are collected for both evaluations to compute the vector cross-correlation coefficient and quantify the equivalence between the strategies (FI campaigns and the proposed one). When the error model is accurate enough to represent hardware faults, it can evaluate complete CNNs.

C. Results and analyses

In the experiments, we use a tool named *PyOpenTCU*, which combines the behavioral operation of the schedulers and general GPU's hierarchy with the instruction-accurate TCU architecture [38], [39]. For the purpose of this work, *PyOpenTCU* is configured with 7 SMs and 28 TCUs.

TABLE III: DNN LAYERS FOR ERROR MODEL VALIDATION.

DNN layer	GEMM shape (A×B×C)
ResNet18 RB1	64×147×12,544
ResNet18 RB2	64×576×12,544
MobileNetV2	3×3×12,544
LeNet5	6×75×576
YoloV5	32×27×102,400

TABLE IV: NUMBER OF FAULTS MODELED AS ERRORS (*bit-flip-masks*) AND COMPRESSED CLUSTERS PER GEMM SEED.

Golden GEMM seed	faults as *bit-flip-masks*	Clusters
100X	7,998 (94.1%)	1,350
200X	7,052 (83%)	1,159
300X	7,740 (91.1%)	1,052
400X	7,602 (89.4%)	939

For the shape-wise error model generation, eight statistical FI campaigns injected $6.8x10^4$ permanent (*stuck-at*) faults on the TCUs of one SM, with the 95% of confidence level and 1% error margin, and observed the corruptions only at the GEMM's outputs. The first 4 FI campaigns resort to GEMM seeds with shapes: **100X** ($100 \times 100 \times 100$), **200X** ($200 \times 200 \times 200$), **300X** ($300 \times 800 \times 300$), and **400X** ($400 \times 1,200 \times 400$) to identify corruptions and define the initial error models. Then, a second set of 4 campaigns performs the error refinement. We use 5 representative convolutional layers with different GEMM shapes for validation purposes (*LeNet5*, *ResNet18-RB1*, *ResNet18-RB2*, *Yolov5*, and *MobileNetv2*), as reported in Table III. Five additional FI campaigns (one per layer) injected around $5x10^3$ faults to generate the reference corruptions for validation against the proposed shape-wise error models. All experiments used a workstation HP Z2G5 with a 20-core Intel i9-10800 CPU and 32 GB of RAM.

A preliminary analysis indicates that a considerable percentage of the evaluated faults in TCUs (around 70% to 90%) produced corruption effects (SDCs).

We refined the initial scalar error models for each fault that causes corruptions (*bit-flip-masks*) during the FI campaigns. Then, we compare the MAEs from the additional FI campaigns (randomly generating new GEMM seeds for each fault) and those obtained through the error models.

Our exploration of different thresholds *(Th)* indicates that a narrow *Th* (i.e., 1) reduces the number of *bit-flip-masks* representing the corruption effects from hardware faults and only groups those errors with high corruption accuracy. However, when *Th* moves from 3 to 10 the total amount of *bit-flip-masks* describing corruptions under acceptable levels increases from about 66% to around 93% on all analyzed GEMM seed shapes under the same order of magnitude. This proves that the refinement can provide acceptable accuracy to represent errors, as reported in Table IV with compression results on each shape-wise error model for *Th*=10.

In some cases, e.g., 100x GEMM seed, the shape-wise error model (*bit-flip-masks*) covers a considerable percentage

of effectively represented errors (up to 94.1%) from hardware faults under an acceptable number of clusters (1,350). Thus, instead of using a conventional FI campaign (7,998 faults), an application-level error evaluation provides the equivalent corruption effects by evaluating only 1,350 errors with our approach. Moreover, the experimental results support the idea that compression can effectively reduce the number of corruption errors to evaluate in large GEMM workloads with a similar impact on their evaluation times (5X to 8X).

Finally, we validated the effectiveness of the shape-wise error models by determining the cross-correlation factor (the higher, the better) between conventional FI campaigns and our error modeling strategy for the DNN layers. The results, illustrated in Figure 5 *(Left)*, show that the GEMM seed's shape is vital for the effective representation of errors at the software level. The results suggest that small-shape GEMMs (100X with correlations from 55% to 92%) are more accurate than large ones (e.g., 400X with correlations from 15% to 52%) in representing errors on all evaluated layers. A comparison of MAEs from the FI campaign and the shape-wise error model for *ResNet18 RB1* layer in Figure 5 *(Right)* shows that up to 92% of the *bit-flip-masks* accurately represented errors. In contrast, large-shape GEMMs are affected by the amount and size of operations in the layers and the GEMM seeds (i.e., a similar amount and size of operations in 100X shape-wise error models and the evaluated layers, while completely different for the 300X and 400X models).

The results indicate that our strategy can represent with a good accuracy the effect of faults and optimize application-level reliability evaluations with performance improvements of up to 225X (from 8.82 h in conventional FIs to 2.26 min using our error modeling strategy with **100X** *bit-flip-masks*).

V. Cross-Layer simulation methodologies in AI accelerators

Deep Learning Accelerators (DLAs) are typically optimized during design-time to meet stringent area, power, and performance requirements. However, these architectural parameters can also impact the propagation of faults through the system, affecting overall dependability. While existing studies have explored the correlation between architectural parameters and SDC errors, they remain in the preliminary stages and lack comprehensive methodologies to evaluate the impact of design choices on system reliability [45] [46]. Simulation-based and emulation-based techniques offer high fault-injection control but often suffer from long simulation times, limiting their applicability to early design phases [47] [48].

In this section, we present a cross-layer simulation methodology designed to evaluate the reliability of DLAs by injecting faults at various levels of abstraction. Our approach leverages a cycle-accurate microarchitectural simulation tool that allows for fault injection in software-visible registers while maintaining the time-awareness of RTL simulations [49]. This enables us to model how architectural parameters influence SDCs and assess their impact on DNN applications in terms of performance and dependability.

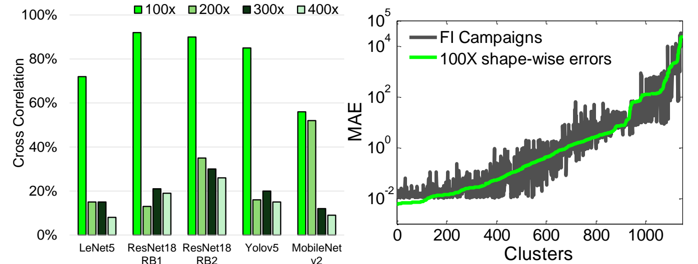


Fig. 5: Cross-correlation on the shape-wise error models for the evaluated layers.

A. Case Study: The NVIDIA Deep-Learning Accelerator

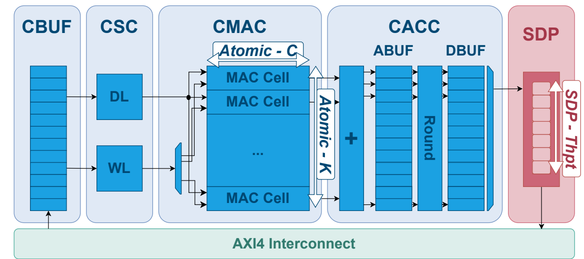


Fig. 6: The NVDLA pipeline as block diagram

The *NVIDIA Deep-Learning Accelerator (NVDLA)* [50] is a domain-specific accelerator designed for 2D convolution operations. It features a five-stage convolution pipeline, comprising key components such as a dedicated memory interface *(CDMA)*, tightly coupled input buffers *(CBUF)*, a control unit for data and weight loading *(CSC)*, a multiply-and-accumulate unit *(CMAC)*, and an accumulation unit *(CACC)*. The pipeline is followed by a reconfigurable post-processing unit *(SDP)*, which supports element-wise operations like activations.

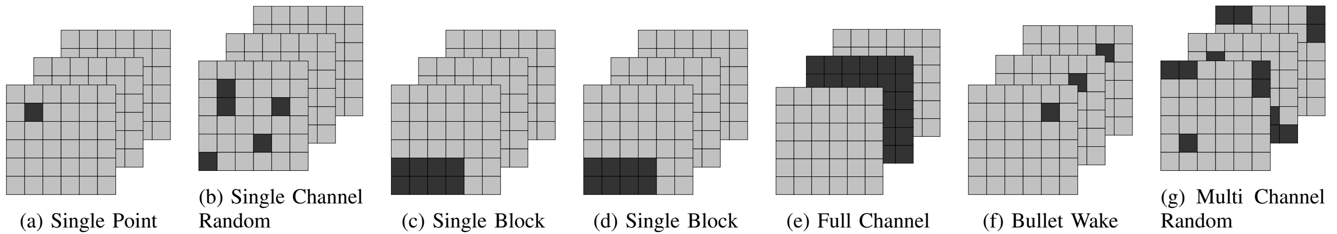
The system's flexibility stems from its configurability. Key parameters include the buffer sizes, the number of input/output channels handled by the CMAC *(Atomic-C, Atomic-K)*, and the precision of integer pipelines (8, 16, or 32 bits). By analyzing different configurations, we assess how hardware choices impact fault propagation and system reliability during deep learning operations.For further details on the NVDLA architecture, refer to our previous work [49].

B. Cross-Layer workflow

To overcome RTL simulation speed, without loosing the time information, we adopt in our workflow a cycle-accurate microarchitectural model extended with hardware information.

The example in Pseudocode 1 shows a possible implementation of the NVDLA CMAC unit, performing vector-to-matrix multiplications of sizes *Atomic-C* $\times$ *Atomic-K*. While Pseudocode 2 presents a possible extension to model it as an array of multipliers, followed by an adder tree, which critical path is reduced by inserting a register barrier in the middle.

```
cmac(weights, inputs, psums):
  for outCh in range(0, Atomic-K):
    for inCh in range(0, Atomic-C):
```



(a) Single Point (b) Single Channel Random (c) Single Block (d) Single Block (e) Full Channel (f) Bullet Wake (g) Multi Channel Random

Fig. 7: Observed error patterns in the output tensor of single DNN layers.

```
4      // Multiply-and-Accumulate
5      psums[outCh] += inputs[inCh] * weights[outCh][
    inCh];
```

Pseudocode 1: "Plain" implementation of NVDLA MAC array.

```
1  cmac(weights, inputs, psums):
2    for outCh in range(0, Atomic-K):
3      // Multiplier Array
4      for inCh in range(0, Atomic-C):
5        mulout[inCh] = inputs[inCh] * weights[inCh];
6      // Adder Tree - Level 1
7      for inCh in range(0, Atomic-C):
8        reg[inCh / regnum] += mulout[inCh];
9      // Adder Tree - Leve 2
10     for rit in range(0, regnum):
11       psums[outCh] += reg[rit];
```

Pseudocode 2: "Extended" implementation of NVDLA MAC array. `regnum` is the number of registers in the middle of the adder tree.

By modeling hardware registers as software-visible variables, we can observe the register content at simulation time and, thus, inject it with faults in the form of bitflips or corruption masks thanks to the support of saboteurs. The latter can be easily built after an accurate RTL description inspection, thus closely matching the hardware behaviour.

Last, to characterize the error models observed in the accelerator, we feed the simulation results to CLASSES [12]. This state-of-the-art tool is composed of an analyser, extracting error models statistics, and by a network error simulator, which replicates the observed error models in order to asses the full-system fault tolerance [45].

C. Error Modeling Results

TABLE V: ANALYSED NVDLA CONFIGURATIONS. BUFFER SIZES ARE NORMALIZED ON THE DATAPATH BITWIDTH.

Configuration	CBUF size	Atomic-C	Atomic-K	CACC size	SDP thpt
nv_small64	131072	8	8	8192	1
nv_small256	262144	32	8	8192	1
nv_medium512	262144	32	16	32768	4
nv_medium1024	262144	32	32	65536	8
nv_large2048	262144	64	32	65536	16

Table V reports the analyzed NVDLA configurations (available at [50]). They majorly differ in MAC array sizes and post-processing unit parallelism. However, buffers don't linearly scale up with the configuration complexity. In particular, the CBUF is constant through most of the tested designs. We performed a SEU injection campaign in the form of bitflips in registers for every combination of designs under test and tested the network layer. The layers were selected from AlexNet and ResNet-18 and trained on CIFAR-10 and CIFAR-100, respectively. Each fault injection campaign consists of 10.000 SEU injections, for a total of 10K simulations for 15 designs and 24 convolutional layers.

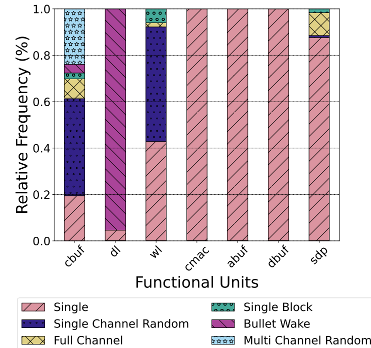


Fig. 8: Relative frequency of error patterns across hardware units in the *nv_small256* configuration, quantized to 16-bit precision.

The observed error models (Figure 7) present in the output tensors belong to six different geometries previously characterized for GPUs [12]. They are referred to as:

- **Single Point:** A single output tensor value is corrupted;
- **Single Channel Random:** Multiple corrupted values in the same channel;
- **Single Block:** Corrupted elements sharing the same channel having contiguous X-Y locations;
- **Full Channel:** An entire channel is corrupted;
- **Bullet Wake:** Multiple corrupted values across different channels sharing the same X-Y coordinates;
- **Multi Channel Random:** Multiple values across different X-Y coordinates and channels.

Figure 8 presents the relative distribution of the observed error models into the nv_small256 configuration running with integer 16-bit precision. *Single Point* errors are by far the most common in the pipeline and are associated with corrupted data that are not reused during the computation. On the other hand, data from the CBUF, DL, or WL easily turns into error models associated with multiple erroneous values in the output tensor.

Full Channel, *Single Block*, and *Single Channel Random* error patterns are always associated to a weight data corruption, where the difference between the three is determined by the error activation probability. An erroneous weight can impact even all the output partial sums associated with a given neuron, but they always share the same output channel. In fact,

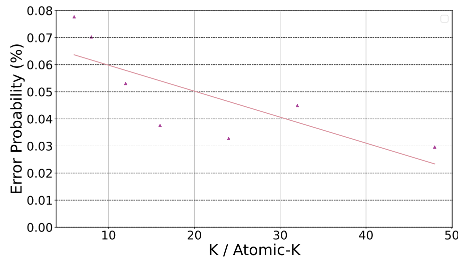


Fig. 9: Probability of output tensor corruption over the number of tiled convolutions (*K* / *Atomic-K*).

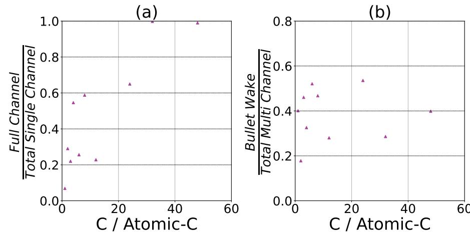


Fig. 10: Relative frequencies of error patterns for faults in the weights (a) and faults in the features(b), against the number of CBUF entries (C / *Atomic-C*).

due to the weight-stationary data reuse policy of NVDLA, we know how a corrupted weight won't be prefetched until the accelerator processes all the corresponding input features, thus impacting multiple X-Y coordinates. Differently, *Multi Channel Random* and *Bullet Wake* error patterns are associated with feature data corruption. In fact, corrupted feature data are broadcasted to multiple neurons, impacting multiple output channels. Thanks to the hardware parallelism of NVDLA, this also suggests how a multi-channel pattern cannot impact more than *Atomic-K* channels at a time. Last, we can also observe a minor presence of *Full-Channel*, *Single Block*, and *Single Channel Random* error patterns in the post-processing unit. They are majorly associated with biased data corruption. Since biases are loaded at the beginning of the execution and are added to all the elements of the same output channel.

The probability that, given an SEU, this translates into any of the observed error patterns is plotted in Figure 9. As can be observed, this exposes a negative trend against the ratio $\frac{K}{\text{Atomic-K}}$, where K is the number of output channels of a layer and *Atomic-K* is the number of hardware neurons. It is easy to observe how this ratio represents the number of hardware convolutions the layer is tiled into. Therefore, since the more the tiles, the smaller they are, with the growth of the $\frac{K}{\text{Atomic-K}}$ ratio, we are computing fewer data together, so the probability that corrupted data is computed with sensitive data decreases.

Second, Figure 10 plots the relative frequency of error patterns associated with weights and feature corruption in the CBUF. As illustrated, there is a growing trend in association with the growth of the $\frac{C}{\text{Atomic-C}}$ ratio, where C is the number of layer's input channels, and *Atomic-C* is the number of hardware input channels of the CMAC. This can be explained if we observe how the hardware execution time is proportional

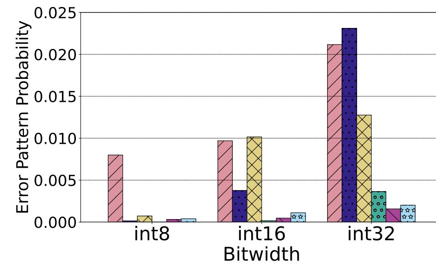


Fig. 11: Error pattern probability in relation to datapath bitwidth. The colors correspond to the error patterns shown in Figure 8.

to the ratio $\frac{C}{\text{Atomic-C}}$. In fact, by increasing the total execution time, it also increases the time window for the CBUF to be exposed. Therefore, it increases the incidence of error patterns associated with CBUF data corruption.

Last, Figure 11 illustrates the probability of different error patterns across the analyzed hardware datapath precisions. It can be observed that the probability of error patterns does not scale uniformly across all datapath precisions. While *Single Point* patterns are by far the most common in 8-bit pipelines, error patterns related to weight data corruption rapidly scale up with larger bit widths. This effect is directly caused by the quantization scheme. Our analysis adopted a symmetric post-training quantization scheme, which quantizes the weights at 8 bits to reduce memory usage. This means that the remaining bits in the pipeline are used for sign bit extension. Faults among the sign bits are way more impactful than bitflips among the information bits. Furthermore, quantization schemes are not designed to account for robustness against faults but to preserve the highest network accuracy with the minimum memory usage. Therefore, pipelines designed to support large data (i.e., 32-bit) computation expose a higher amount of sign bits, and thus a higher probability to generate error patterns different from *Single Point*.

It is possible to observe how different error patterns expose different impacts on the final application [12], [45]. In particular, while *Single Point* errors are the most common, other patterns may be way more dangerous (in particular, *Bullet Wake* patterns). As Figure 9 and Figure 10 illustrate, the relationship between DNN hyper-parameters and DLA hardware parameters has a significant impact on the type of observed error patterns and the probability with which they appear, opening the door for future reliability-driven hardware configuration space explorations.

VI. Conclusions

In this paper we first assessed the effects of faults induced by radiation on large ViTs for image classification. While ViT models provide high accuracy, we observed in our experiments a high percentage of critical SDCs (misclassifications). We then designed a fault tolerance approach based on value range restriction to reduce/remove the critical SDCs. Our approach combines analysis from beam experiments and data from software fault simulation. Furthermore, we analyzed the effectiveness of multi-abstraction assessments to characterize

the impact of hardware faults in TCUs and propose software error models able to preserve accuracy while significantly speeding up reliability evaluation. Finally, we highlighted how different hardware parameters in an NVDLA core can significantly change the observed error models and their occurrence, depending on the target DNN model. Our work opens future perspectives in investigating performance-dependability-cost tradeoffs.

References

[1] A. Khan *et al.*, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, pp. 5455–5516, 2020.

[2] L. Alzubaidi *et al.*, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, pp. 1–74, 2021.

[3] ISO 26262-5, "Road vehicles — functional safety — part 5: Product development at the hardware level," pp. 1–90, 2018.

[4] ISO/IEC 22989, "Information technology — artificial intelligence — artificial intelligence concepts and terminology," pp. 1–60, 2022.

[5] I. Hill *et al.*, "Cmos reliability from past to future: A survey of requirements, trends, and prediction methods," *IEEE Trans. Device Mater. Reliab.*, vol. 22, no. 1, pp. 1–18, 2022.

[6] J. D. Guerrero Balaguera *et al.*, "Understanding the effects of permanent faults in gpu's parallelism management and control units," in *Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC'23)*, 2023.

[7] J. Rajski *et al.*, "The future of design for test and silicon lifecycle management," *IEEE Design & Test*, pp. 1–1, 2023.

[8] IEEE, "The international roadmap for devices and systems: 2022," in *Institute of Electrical and Electronics Engineers (IEEE)*, 2022.

[9] H. D. Dixit *et al.*, "Silent data corruptions at scale," 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

[10] P. H. Hochschild *et al.*, "Cores that don't count," in *Workshop on Hot Topics in Operating Systems*, 2021, p. 9–16.

[11] G. Papadimitriou and D. Gizopoulos, "Silent data corruptions: Microarchitectural perspectives," *IEEE on Trans. Comput.*, vol. 72, no. 11, pp. 3072–3085, 2023.

[12] C. Bolchini *et al.*, "Fast and accurate error simulation for cnns against soft errors," *IEEE Trans. Comput.*, vol. 72, no. 4, p. 984–997, apr 2023.

[13] 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, 2011.

[14] J. E. R. Condia *et al.*, "A multi-level approach to evaluate the impact of gpu permanent faults on cnn's reliability," in *EEE Int. Test Conf. (TC)*, 2022, pp. 278–287.

[15] K. Ito *et al.*, "Analyzing due errors on gpus with neutron irradiation test and fault injection to control flow," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 8, pp. 1668–1674, 2021.

[16] J. M. Badia *et al.*, "Comparison of parallel implementation strategies in gpu-accelerated system-on-chip under proton irradiation," *IEEE Trans. Nucl. Sci.*, pp. 1–1, 2021.

[17] M. B. Sullivan *et al.*, "Characterizing and mitigating soft errors in gpu dram," in *54th Ann. IEEE/ACM Int. Symp. on Microarchitecture*, 2021, pp. 1–10.

[18] P. R. Bodmann *et al.*, "Soft error effects on arm microprocessors: Early estimations versus chip measurements," *IEEE on Trans. Comput.*, vol. 71, no. 10, pp. 2358–2369, 2022.

[19] J. E. R. Condia *et al.*, "Flexgripplus: An improved gpgpu model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[20] A. Chatzidimitriou *et al.*, "RT level vs. microarchitecture-level reliability assessment: Case study on ARM(r) cortex(r)-a9 CPU," in *DSN Workshop*, 2017.

[21] C. Constantinescu *et al.*, "Error injection-based study of soft error propagation in amd bulldozer microprocessor module," in *DSN*, 2012.

[22] S. S. Mukherjee *et al.*, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *MICRO*, 2003.

[23] A. R. Anwer *et al.*, "Gpu-trident: Efficient modeling of error propagation in gpu programs," in *SC*, 2020.

[24] T. Tsai *et al.*, "NVBitFI: Dynamic Fault Injection for GPUs," in *Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[25] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from Architectural Vulnerability." IEEE HPCA, 2009.

[26] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *IEEE ISCA*. IEEE ISCA, 2021, p. 902–915.

[27] F. F. d. Santos *et al.*, "Characterizing a neutron-induced fault model for deep neural networks," *IEEE Trans. Nucl. Sci.*, vol. 70, no. 4, pp. 370–380, 2023.

[28] 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, no. 1, p. 012037, may 2018.

[29] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." 9th Int. Conf. on Learning Representations (ICLR), 2021.

[30] Y. Fang *et al.*, "Eva-02: A visual representation for neon genesis," *arxiv*, 2023.

[31] Z. Liu *et al.*, "Swin transformer v2: Scaling up capacity and resolution." IEEE IEEE/CVF Computer Vision and Pattern Recognition Conf. (CVPR), 2022, pp. 12 009–12 019.

[32] Z. Tu *et al.*, "MaxViT: Multi-axis vision transformer." ECCV, 2022, pp. 459–479.

[33] R. Wightman, "Huggingface," huggingface.co/timm.

[34] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *IEEE CVPR*. IEEE CVPR, 2009, pp. 248–255.

[35] F. F. d. Santos *et al.*, "Demystifying gpu reliability: Comparing and combining beam experiments, fault simulation, and profiling," in *IPDPS*, 2021.

[36] G. Gavarini *et al.*, "Evaluation and mitigation of faults affecting swin transformers." IEEE 29th Int. Symp. on On-Line Testing and Robust System Design (IOLTS), 2023.

[37] Z. Chen *et al.*, "A Low-cost Fault Corrector for Deep Neural Networks through Range Restriction." IEEE/IFIP DSN, 6 2021.

[38] R. L. Sierra *et al.*, "Analyzing the impact of different real number formats on the structural reliability of tcus in gpus," in *IFIP/IEEE 31st Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, 2023, pp. 1–6.

[39] R. Limas Sierra *et al.*, "Exploring hardware fault impacts on different real number representations of the structural resilience of tcus in gpus," *Electronics*, vol. 13, no. 3, 2024.

[40] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," 2017.

[41] V. Thakkar *et al.*, "CUTLASS," 2023.

[42] A. Ruospo *et al.*, "A pipelined multi-level fault injector for deep neural networks," in *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020, pp. 1–6.

[43] R. Limas Sierra *et al.*, "Analyzing the impact of scheduling policies on the reliability of gpus running cnn operations," in *42nd IEEE VLSI Test Symp. (VTS 2024)*, 2024, pp. 1–7.

[44] K. Q. Ye, "Indicator function and its application in two-level factorial designs," *The Annals of Statistics*, vol. 31, no. 3, pp. 984 – 994, 2003.

[45] A. Veronesi *et al.*, "Cross-layer reliability analysis of nvdla accelerators: Exploring the configuration space," in *IEEE European Test Symp. (ETS)*, 2024, pp. 1–6.

[46] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *55th ACM/ESDA/IEEE Design Automation Conf. (DAC)*, 2018, pp. 1–6.

[47] S. Pappalardo *et al.*, "A fault injection framework for ai hardware accelerators," in *24th Latin American Test Symp. (LATS)*, 2023, pp. 1–6.

[48] X. Feng *et al.*, "Runtime fault injection detection for fpga-based dnn execution using siamese path verification," in *2021 Design, Automation & Test in Europe Conf. & Exhibit. (DATE)*, 2021, pp. 786–789.

[49] A. Veronesi *et al.*, "Exploring software models for the resilience analysis of deep learning accelerators: the nvdla case study," in *Int. Symp. on Design and Diagnostics of Electronic Circuits and Systems*, 2022, pp. 142–147.

[50] "The nvidia deep-learning accelerator." [Online]. Available: www.nvdla.org