Evaluating Different Fault Injection Abstractions on the Assessment of DNN SW Hardening Strategies

Evaluating Different Fault Injection Abstractions on the Assessment of DNN SW Hardening Strategies

Giuseppe Esposito, Juan-David Guerrero-Balaguera, Josie E. Rodriguez Condia, Matteo Sonza Reorda
Politecnico di Torino - Department of Control and Computer Engineering (DAUIN), Turin, Italy
{giuseppe.esposito, juan.guerrero, josie.rodriguez, matteo.sonzareorda}@polito.it

***Abstract*—[1] The reliability of Neural Networks has gained significant attention, prompting efforts to develop SW-based hardening techniques for safety-critical scenarios. However, evaluating hardening techniques using application-level fault injection (FI) strategies, which are commonly hardware-agnostic, may yield misleading results. This study for the first time compares two FI approaches (at the application level (APP) and instruction level (ISA)) to evaluate deep neural network SW hardening strategies. Results show that injecting permanent faults at ISA (a more detailed abstraction level than APP) changes completely the ranking of SW hardening techniques, in terms of both reliability and accuracy. These results highlight the relevance of using an adequate analysis abstraction for evaluating such techniques.**

***Index Terms*—Deep Neural Networks, Fault tolerance, SW hardening, Reliability assessment, Safety, Artificial Intelligence**

I. Introduction

Today, Artificial Intelligence (AI) influences many areas of technology. It is applied in IoT devices, like robots, drones, and automotive systems, as well as chatbots across various domains powered by Large Language Models. The complex nature of AI requires significant computational resources, usually provided by powerful hardware systems supported by specialized AI accelerators, including *Graphics Processing Units* (GPUs) [1]–[3]. In fact, the development of AI accelerators (e.g., GPUs) is boosted by the continuous evolution of semiconductor technologies. Unfortunately, as the semiconductor technologies scale down (7nm and below), devices exhibit increased susceptibility to faults, compromising system reliability. Some studies indicate that factors, such as process variation, wear out, and harsh environments accelerate device aging and degradation, resulting in hardware faults that manifest as *Silent Data Errors* (SDEs) [4]–[6] at the application level. In AI applications (e.g., *Deep Neural Networks*, or DNNs), faults affecting the hardware can lead to computational errors/failures, reducing their performance and compromising their correct behavior.

Several works aimed at reducing the impact of hardware faults on DNNs by adopting software-based fault mitigation strategies acting on the DNN architecture [7]. These fault countermeasures seek to increase the robustness or, in other cases, complement well-known approaches, such as error-detection/-correction mechanisms (i.e., ECCs) [8], [9]. In this work, we target only software-based hardening techniques implemented at the application level, neglecting lower-level software-based hardening strategies (e.g., Algorithmic-Based Fault Tolerance techniques [10]). Among the software-based hardening approaches for DNNs, the most common ones resort to activation bounding with or without retraining (e.g., *Ranger* [11], *Adaptive Clipper* [12], *Swap ReLU6* [7] or *Median filter* [13]), redundancy (e.g., TMR) or hardware-aware pruning for TPU architectures [14]. The experiments supporting the effectiveness of these methods generally resort to Fault Injection (FI) campaigns that inject permanent or transient faults to validate the different techniques.

These FI campaigns can be carried out at different levels of abstraction, such as *i) physical-based*, *ii) hardware-based*, *iii) Instruction/ISA-based*, and *iv) Application/APP-based* [15]. The first three categories are hardware aware and can provide realistic results; nonetheless, they require specialized tools and hardware descriptions, leading to costly, nonscalable, and prohibitive evaluation times, especially when dealing with DNN workloads [16]. In contrast, application/APP-based FI strategies are more flexible and faster than other approaches. Consequently, the reliability evaluation of DNNs (and their SW-based hardening strategies) has widely adopted FI evaluations at the APP level (i.e., corrupting DNN weights/feature maps). Unfortunately, evaluations at the APP level are hardware-agnostic, so they can hardly describe the actual effect of faults occurring in the underlying hardware, and this affects the accuracy of the evaluations. As an alternative solution, hardware injection through program transformation (HITPT) is a SW-based FI technique that corrupts at the instruction level (ISA) the execution of any parallel program, including DNNs. To do so, HITPT modifies the assembly code of the DNN execution to inject a fault in the target component. HITPT allows modeling the effect of permanent faults in a real device more realistically, which is suitable for effective reliability evaluation of complex applications, such as DNN workloads and SW hardening strategies. Moreover, HITPT is faster than evaluation-based FI approaches and provides equivalent insights into faults affecting the GPUs as the FI occurs closer to the internal HW structures than application-level approaches.

In this work, we present, for the first time, a comparative evaluation made by resorting to different FI abstractions to study the effectiveness of several DNN SW hardening strategies w.r.t. permanent faults. The comparison is made by resorting to custom tools: a customized version of PytorchFI [17], which allows the corruption of weights and feature maps of a DNN, and an adapted version of NVBitFI, which injects permanent faults (i.e., stuck-at faults) in the register files and the input/outputs of the functional units of a GPU. We considered 3 general-purpose DNNs (*Lenet5*, *MobilenetV2*, and *Resnet18*)

[1]This work has been supported by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing.

979-8-3315-2916-1/24/$31.00 ©2024 IEEE

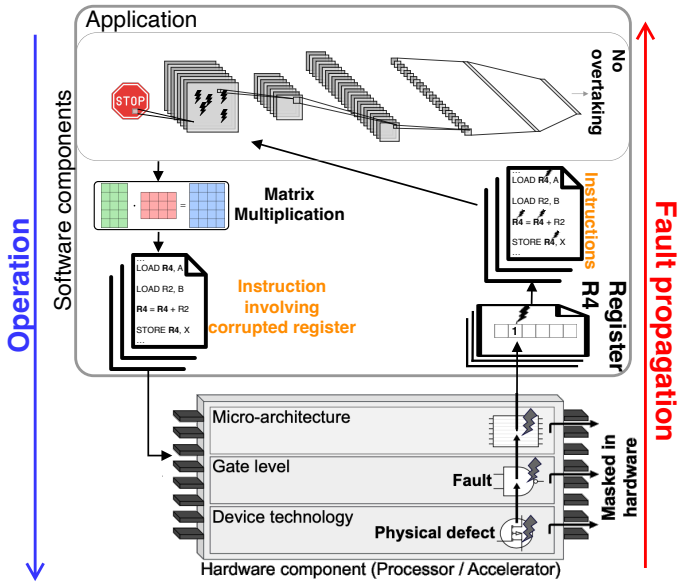

Fig. 1. Permanent fault propagation from the hardware to the application possibly generating an error.

in the original and hardened versions, resorting to 3 different state-of-the-art SW hardening strategies (*Swap ReLU6* [7], *Ranger* [11] and *Adaptive Clipper* [12]). The results show that results produced by APP-level FI are significantly different than those from ISA-level FI. More in general, the adoption of a hardware-aware FI technique (thereby providing more accurate results) may result in a complete change of the ranking of the SW hardening techniques w.r.t. the ranking obtained resorting to APP-level FI.

The paper is structured as follows: Section II outlines the possible reliability evaluation strategies. Section III describes the experimental environment and the supported Fault Models. The experimental setup and the results are reported in Section IV. Section V concludes the paper and lists possible future works.

II. Related works

Over recent years, several studies have addressed the challenge of effectively assessing DNN robustness through FI techniques, focusing on evaluation strategies that can work at different levels of abstraction such as *i)* APP, *ii)* ISA, *iii)* microarchitectural-level and *iv)* Register Transfer Level (RTL). The *APP* FI campaigns involve the perturbation of the parameters of the DNN or of the intermediate outputs. For example, authors of [18], [19] conducted 2 different reliability-oriented analyses evaluating the best DNN weights representation format and the best DNN architecture, respectively, by performing random bit-flips on weight parameters. The approach simulates errors caused by faults without accounting for how data and operations are assigned to the hardware. Although the high customization degree and low execution time represent advantages in terms of controllability and feasibility of the experiments, a key drawback of APP FIs is the hardware-agnostic approach that is adopted [16].

ISA FIs provide a better trade-off between evaluation time and result accuracy. ISA FI can resort to the HITPT strategy, which allows for injecting faults in kernel instructions of parallel executions [20]. For transient faults on GPUs, NVBitFI [20] and SASSIFI [21] are state-of-the-art frameworks based on the corruption of the binary code and the PTX code, respectively, to inject faults. DNNs reliability assessment through ISA-level FIs has been explored in [22], [23] where the authors extended the use of this approach to permanent faults affecting registers and functional units. Moreover, in [24] the authors identified the input values that activate faults injected by instrumenting the GPU assembly code. Lastly, in [25] the authors conducted an analysis of two ARM ISAs and two microarchitectures for each ISA to quantify errors in architecture-level and SW-level vulnerability evaluation.

Fine-grained reliability evaluations (at the *micro-architectural level*) of DNNs on hardware like GPUs are highly detailed but extremely time-consuming. For example, GUFI [26] performs FIs on a GPU microarchitecture simulator, corrupting hardware components at random execution cycles rather than altering the application source/binary code. Despite higher experimental accuracy, GUFI's long evaluation times make it impractical for complex applications like DNNs.

RTL abstraction, which describes circuit behavior through data transfers between registers and logical gates, allows for more detailed hardware characterization. For example, RTL FIs were used in [27] to identify critical GPU sites and implement selective hardening for improved reliability. However, the high resource consumption, complexity, and huge development time make a full reliability assessment of DNNs impractical at this level (or lower).

To summarize, most of the current DNN reliability evaluation approaches adopt the APP FI due to its easy applicability and high speed, but neglect the result inaccuracy. To the best of our knowledge, no existing study has analyzed the effect of different abstraction levels when evaluating SW hardening strategies for DNNs. This work for the first time compares the results coming from ISA FIs and APP FIs when SW hardening strategies are applied to DNNs.

III. Evaluation methodology

This section describes the adopted strategy to evaluate and analyze the impacts of two different abstraction levels (ISA and APP) on the reliability assessment of SW-hardening techniques for DNNs. In particular, we evaluate permanent (stuck-at) faults during the inference of DNNs on GPUs. The first abstraction (ISA or *CUDA instruction-level*) focuses on the HITPT mechanisms to corrupt the program, while APP evaluation induces errors by directly corrupting DNN parameters to represent errors from hardware faults. Based on the error induced on the final DNN output, the fault classification allowed us to outline fault effects for each hardened application, highlighting its resiliency in front of different abstraction levels. Moreover, the performed FI experiments allow assessing the severity of the injected faults and the consequent induced error.

A. Hardware-Injection Through Program Transformation (HITPT)

As illustrated in 1, when an operation is executed, defects might be activated and propagated as a hardware fault (e.g., corrupting the logic state on a gate) across the micro-architecture of the system during the application's execution and cause failures/errors in the system. More in detail, corrupted logic states can impact multiple operations of assembly instructions (e.g., corrupted assembly code workflow) and produce incorrect final application outputs or impact their parameters.

To inject faults, in this work, we employ an adapted version of NVBitFI, which implements the HITPT approach to model, at the SW level, the fine-grain effect of faults in the data path structures of a GPU. In particular, the framework injects permanent (*stuck-at*) faults in the inputs/outputs of Functional Units (FUs) and on Register File (RFs) destinations for the kernel instructions. Then, we perform exhaustive FI campaigns on all used registers per RF and FUs (separately) for a targeted Thread, which is executed on a specific *Streaming Multiprocessor* (SM) inside a GPU.

B. Application-level FI

The fault injector used for this type of abstraction level is based on PyTorchFI [17], which allows the corruption of DNN components on the parameter tensor (*weights*) or in the intermediate output tensor (*neuron's output*).

For each DNN under testing, we have performed 2 FI campaigns. In the first one, we injected Single Bit-Flips (SBFs) for each DNN run through statistical FI [28] targeting weight parameters. In the second campaign, we injected Multiple Bit-Flips (MBFs) according to a given BER. In the first part of the campaign, we targeted weight parameters and the injected faults are obtained by extracting a BER fraction out of the available bit space. After sampling a sufficient number of faults, using the PytorchFI forward hook function, we can perturb the weights before the DNN inference. While the DNN weights are fixed, the neuron's output changes depending on the target layer's input *Feature Map* (FM). Therefore, in the second campaign, we targeted neurons' outputs through PytorchFI which allows to corrupt DNN intermediate output during inference. We used the framework presented in [29] where the error list contains the desired tail size the Block Error Rate ($BlER$), the Neuron Error Rate (NER), and the bit location. All those fault features, define the FM portion to corrupt along with the specific bit locations.

C. Evaluation process

For each level of FI abstraction, injected faults are classified into different categories depending on the severity of their impact on the application performance. Specifically, the recorded accuracies of the fault-free and corrupted DNNs are evaluated so that the framework can compute the *Relative Accuracy Degradation* (RAD) to quantify the DNN accuracy degradation. Mathematically, RAD is defined as $\frac{ACC_{fault-free}-ACC_{faulty}}{ACC_{fault-free}}$ where $ACC_{fault-free}$ represents the golden accuracy and ACC_{faulty} is the accuracy of the corrupted DNN. The corresponding prediction confidences are compared with the ones generated in the fault-free scenario and the fault classifier assigns the label *Masked* if they match (because the injected error does not change the inference output), or *Safe Silent Data Corruption (Safe-SDC)* (when the fault does not change the prediction), or *Critical Silent Data Corruption (Critical-SDC)* (when the fault does change the prediction). In case the fault has produced a system hang or crash, it is classified as *Detectable Unrecoverable Error (DUE)* that is generally due to memory access violation or memory misalignment violation.

In this work, we first perform a preliminary assessment comparing the fault distribution of the campaigns conducted at ISA (injecting single stuck-at faults) and APP (performing SBFs) abstraction levels to compare the effect of the fault/error injected in 2 different levels of abstraction.

Nevertheless, from a SW standpoint, the propagation of the error, induced by corrupting assembly operations, up to the DNN execution can turn into the corruption of several intermediate outputs/parameters. Therefore, to make a fair comparison, we have computed the error induced by single stuck-at faults at ISA and we induced the same error at APP to make a fair comparison between 2 abstractions. To do so, after each ISA FI, we count and preserve the number of times that the fault is excited (i.e., the target bit assumes a different polarity, either 0 or 1, w.r.t. the type of injected stuck-at fault) by the CUDA kernel execution. Consequently, by dividing by the total number of times the register is used, we have computed the induced Bit Error Rate (BER). After performing ISA FI campaigns, the results obtained were used to define the range [BER_{min},BER_{max}]. Subsequently, 10 values were sampled within this range and employed to perform MBFs at APP. After running the simulations, we compared the impact (in terms of accuracy) of FI campaigns at different abstraction levels. This enabled a fair comparison in the accuracy of the benchmarks under identical error activation rates.

IV. Experimental results

The first set of FI campaigns resorts to ISA FI and focuses on permanent (*stuck-at*) faults arising on all registers in the RFs and all FU cores from the first SM and targeting a given Thread ID. In detail, we injected all possible single faults in registers (totaling 10,496 runs) and all possible single faults in the inputs and outputs of the FUs (1,536 runs).

For the second FI campaign (at the APP level), we calculated the optimal number of corruptions (i.e., single stuck-at faults are injected in the weights as presented in [18]) such that experiment confidence level =95% and error margin=0.5. When multiple bit-flips for each evaluation are injected, the percentage of faults (i.e., BER) ranges within $[1 \cdot 10^{-6}, 6 \cdot 10^{-4}]$.

The ISA FI experiments were performed on a workstation HP Z2 G5 with an Intel Core i9-10800 CPU with 20 cores, 32 GB of RAM, and equipped with one NVIDIA GPU RTX 3060TI, requiring around 96h ($\approx 60h$ for Regs FIs and $\approx 36h$ for FUs FIs). On the other hand, the APP FIs used a 6-node cluster system with 2 Intel 16-core Xeon Scalable Processors Gold 6130 2.10 GHz per node and equipped with 6

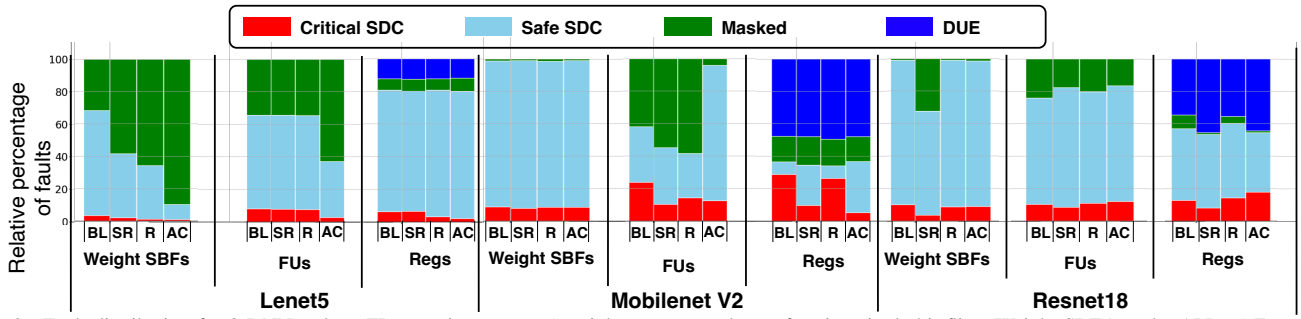

Fig. 2. Fault distribution for 3 DNNs where FI campaigns target *i)* weight parameters by performing single bit-flips (Weight SBFs) at the APP, *ii)* Functional Units (FUs) at ISA and *iii)* Registers (Regs) at ISA. Specifically, the proportion of faults is indicated in relation to each compromised component. Moreover, each bar corresponds to the baseline (BL) unhardened model, Swap ReLU6 (SR), Ranger (R) and Adaptive Clipper (AC).

NVIDIA V100 SXM2 and 32 GB of RAM, requiring around 3h ($\approx 1h$ for SBFs FIs and $\approx 2h$ for MBFs FIs). For both FI campaigns (ISA and APP), we considered three SW-based hardening techniques (*Ranger* [11], *Swap ReLU6* [7], and *Adaptive Clipper* [12]) applied on *LeNet5* (trained on Mnist), *Mobilenet V2*, and *Resnet18* (both trained on Cifar10) and image classifiers, as representative DNN workloads.

A. Error distribution impacts from both evaluation abstractions

Fig. 2 outlines the relative fault distribution stemming from the APP (*weight SBFs*) and ISA (*FUs* and *Regs*) evaluations on the three analyzed hardened-free DNNs (*baseline* or **BL**) and their hardened versions (*Swap ReLU6* or **SR**, *Ranger* or **R**, and *Adaptive Clipper* or **AC**).

The experimental results show that, according to the reliability assessment abstraction (APP or ISA), the DNN resiliency can quantitatively differ. In general, we observed that depending on the FI abstraction level the effectiveness of a hardening mechanism may change significantly. In two of the three analyzed DNNs (Lenet5 and Mobilenet V2), all FI campaigns show a clear reduction in Critical SDCs (from 28.60% to 4.85%). On the other side, moving to Resnet18, we notice that APP FI results show a negligible impact by the hardening techniques, while ISA FI results show that both R and AC hardening techniques produce an increase in the percentage of Critical SDCs. We can also notice that APP FI never produces any DUE, while ISA FI does when we inject into Regs. In this case, DUEs are mainly due to the corruption of instruction destination registers, leading to illegal memory accesses. We can also see that ISA FI strategies significantly change (up to $3\times$) the percentage of Critical SDCs produced by the baseline DNN inference (BL) w.r.t. the results of APP FI when Mobilenet V2 is evaluated. On the other side, for the baseline Lenet5 the Critical SDC percentage increases by almost $2\times$ depending on the FI approach. For Resnet18 the same percentage remains constant due to the inherent and higher redundancy of this DNN model.

Our results indicate that the percentage of Critical SDCs produced by each DNN may change significantly (both in the BL and in any of the hardened versions) depending on the usage of Regs and FUs done by each DNN. On the one hand, DNNs actively using registers (*Mobilenet V2* and *Resnet18*) show a higher percentage of Critical SDCs w.r.t. faults impacting functional units (around 2%). On the other hand, the results on *Lenet5*, with reduced utilization of accessible registers, showed a more resilient behavior to corruptions on the registers (4.41% of critical SDCs) than on functional units (6.42% SDCs).

These results demonstrate that high-level FI approaches (i.e., APP FIs) tend to overestimate the effectiveness of SW hardening techniques in terms of Critical SDCs. In fact, APP FIs report a limited percentage of Critical SDCs (no more than 10%). In contrast, ISA FIs report a much higher percentage of Critical SDCs (up to 31% in the execution of more complex DNNs, such as *Mobilenet V2* and *Resnet18*). Moreover, APP FIs completely neglect the occurrence of DUEs, which may amount to up to 47.5% in some cases (faults in Regs in MobilNet).

Furthermore, it can be noticed that when weight parameters are corrupted with single stuck-at faults, all SW hardening strategies provide beneficial effects in the Lenet5 architecture, increasing the percentage of Masked faults from about 30% up to about 90%. Specifically, Adaptive Clipper outperforms the other techniques when it is implemented on Lenet5.

Interestingly, Adaptive Clipper reaches better performance when FUs are corrupted, decreasing the Critical faults by 5.27% w.r.t. the baseline DNN.

Adaptive Clipper increases the percentage of Masked faults in Lenet5 by fine-tuning weights and bounding activation functions but is less effective when Registers are corrupted only improving the baseline Masked faults by 1.48%. On the other hand, Mobilenet V2, with built-in range restrictions, shows lower benefits from Adaptive Clipper due to its inherent output variability and architecture in weights and FU corruptions. This can be evinced from the increased Safe SDCs (of 39.10%) and decreased Masked faults (of 37.70%) in front of a decrease of Critical SDCs (of 29.43%) during FUs FI evaluations. From the results obtained from Regs evaluations, the decrease by 23.5% of Critical SDCs suggests improved performance. Nevertheless, there is a slight increase in DUEs (0.10%) compared to the baseline, indicating the types of errors the hardening technique can mask. Ranger and Swap ReLU6 show better performance by slightly reducing faults with critical effects to 14.25% and 10.22%, respectively, when FUs are corrupted. For Resnet18, APP evaluations show that Swap ReLU6 improves performance by reducing the Critical SDCs by 2%, while Ranger maintains it around 10%. Adaptive Clipper, however, is less effective, increasing Critical SDCs by 5.2%. Resnet18 also experiences a significant rise in DUEs with Swap ReLU6 and Adaptive

Clipper, by 10.09% and 11.07%, respectively. Swap ReLU6 reduces Critical SDCs to 7.95%, indicating a lower error probability compared to the baseline and other strategies. In contrast, Adaptive Clipper does not improve but indeed degrades the original reliability, with Critical faults either remaining constant or increasing. This behavior is reflected in all FI results, where the reported percentage of Critical faults is kept constant (to 8.87% in weights parameters corruption) or increased (of 2.16% in FUs corruption). Our experiments indicate that the evaluation abstraction quantitatively affects the effectiveness of a hardening strategy (from 3% to 0.4% of the faults evaluated on the selected workloads). The distribution of critical faults across hardening strategies shows the same trend only for Regs and FUs, revealing different Critical SDC coverage rankings across abstraction levels. Under Regs and FUs corruptions, we can notice that Adaptive Clipper outperforms the other hardening strategies, decreasing the baseline Critical fault percentage up to 29.43% when it is implemented on Mobilenet V2. On the other hand, APP FI highlights that Swap ReLU6 is the best-performing technique in terms of Critical fault decrease of 0.4%. Moreover, the presence of DUEs is shown only by injecting in Regs.

As a conclusion, the experimental results suggest that APP FIs provide misleading results when assessing DNNs and the corresponding SW hardening techniques. Instead, ISA FIs, which support a more detailed representation of hardware faults and their effects, do provide quite different results, significantly changing the hardening techniques ranking. Since ISA FIs are closer to the underlying hardware architecture, these results are indeed closer to the real ones.

B. Impact of faults on DNN accuracy

In order to make a fair comparison of the results obtained from single stuck-at faults injected at ISA and APP, the BER has been computed after performing ISA FIs according to the number of times that the injected fault is activated. The evaluations at ISA proved that the generated BER was within the range of [0,10^{-5}]. To ensure comparability, APP FI campaigns were conducted with various BERs within the specified range. This approach allowed us to evaluate the accuracy of the applications across different abstraction levels.

Table I reports the DNN accuracy *i)* when Neurons and Weights are corrupted at the APP by multiple bit-flips (that represent the corruption effect from individual stuck-at faults) with a BER assuming values within the 2 selected bins ([0,10^{-6}), [10^{-6}, 10^{-5}]) and *ii)* when single stuck-at faults affect FUs and Regs generating a BER value, which represents the activation of the error (BER, DNN level) on the columns.

In our evaluations, we observed that single stuck-at faults in RFs never produced BER errors in the range [10^{-6}, 10^{-5}]. The error induced at the SW level with a high rate generates many errors that make the GPU hang or crush (i.e., DUEs).

From a DNN's architecture standpoint, Lenet5 demonstrates higher accuracy resiliency to both ISA and APP evaluation abstractions with a maximum degradation of 22.88% w.r.t. the golden accuracy, thanks to its lower complexity. On the contrary, more elaborated DNNs, such as Mobilenet V2 and Resnet18, report $2\times$ of accuracy degradation (46.96% and 42.16%, respectively), w.r.t. the golden accuracy.

More in detail, the most aggressive APP evaluation (i.e., when BER =10^{-5}) of Resnet18 and Lenet5 show reduced accuracy (never going above 30.64% and 75.12% w.r.t. the performance reported when single stuck-at faults at ISA have generated the same BER). Specifically, in the latter case, the reported accuracy never reaches more than 42.53% and 86.20%. On the other hand, the hardening strategies can raise those accuracy values to 80.37% for Resnet18 and to 95.01% when faults are injected at the DNN level. On the contrary, the reported accuracy of Mobilenet V2 varies within a larger range when BER is set to 10^{-5} (i.e., [51.28%, 83.63%]) than ISA FIs, where the accuracy never goes above 77.9%.

Overall, we can observe that Adaptive Clipper achieves good performance only when it is implemented on Lenet5, reaching 93.96% when neurons' outputs are corrupted and 95.01% thanks to its light model size IV-A. Similarly, the implemented SW-hardening strategy outperforms the others when faults are injected in FUs with a 91.63% of accuracy. On the other hand, when the error induced by register corruption is around 10^{-6}, the hardening of most SW-hardening strategies is neglected except for Ranger, which maintains the DNN accuracy to the baseline (98%). Overall, Ranger performs well in most evaluations, especially when faults in neurons and weights impact Resnet18 by degrading the baseline model accuracy in only 6.42% and 7.26%, respectively. Nonetheless, during Resnet18 inference, under corrupted registers, none of the hardening strategies improves baseline DNN accuracy of 87.08%. While Swap ReLU6 shows an accuracy of 71.70% when FUs are affected by permanent faults.

Interestingly, in Mobilenet V2, Ranger is the most effective SW-hardening technique among all others in terms of accuracy degradation, when considering neurons (with 13.2%), Regs (with 3.32%), and FUs (with 12.81%) corruption. This means that the fine-grained check at the convolutional block level performed by Ranger helps to fix the error induced by the corruption of the mentioned components that generate entries in the intermediate FMs that are outside of the valid range.

In conclusion, different abstraction levels yield different outcomes in FI campaigns. For example, Swap ReLU6 is ineffective under ISA FI but outperforms others in APP FI on Mobilenet V2. This proves that different FI strategies may provide significantly different results about the impact of hardening techniques on the DNN accuracy. As a typical example, ISA FI with 10^{-5} BER shows that hardened models worsen the accuracy w.r.t. the baseline, with up to a 24.74% degradation in Resnet18.

V. Conclusions

In this work, we evaluated and analyzed the effects of permanent faults at 2 abstraction levels (APP and ISA). We considered 3 SW hardening techniques (Ranger, Adaptive Clipper, and Swap ReLU6).

TABLE I

ACCURACY FOR THE BASELINE DNNS AND THEIR HARDENED VERSIONS. THE SIMULATION RESULTS ARE OBTAINED BY CORRUPTING DNNS (BER, APP-LEVEL) AND BASIC INSTRUCTIONS (SINGLE STUCK-AT, ISA-LEVEL). A COLOR SCALE IS USED TO RANK THE RESULTS PROVIDED BY EACH FI CAMPAIGN ON THE DIFFERENT APPLICATIONS. RED, ORANGE, YELLOW AND GREEN COLOR INDICATE THE HIGHEST TO THE LOWEST ACCURATE RESULT FOR EACH SCENARIO, RESPECTIVELY.

	Fault model			**BER (APP-level)**					**Single stuck-at (ISA-level)**			
	Hardening Strategy			Baseline	Adaptive Clipper	Swap ReLU6	Ranger		Baseline	Adaptive Clipper	Swap ReLU6	Ranger
Mobilenet V2	**Golden accuracy**			81.28%	76.40%	82.46%	81.28%		81.28%	84.13%	72.80%	81.28%
	Neurons	**BER**	10^{-6}	71.24%	55.63%	66.20%	71.60%	**Regs**	72.99%	72.18%	71.59%	77.96%
			10^{-5}	67.36%	51.28%	58.66%	68.08%		-	-	-	-
	Weights		10^{-6}	76.44%	73.75%	84.12%	76.88%	**FUs**	58.34%	66.24%	62.62%	75.15%
			10^{-5}	66.56%	72.45%	81.63%	69.37%		34.32%	49.11%	60.63%	68.47%
Resnet18	**Golden accuracy**			88.14%	84.13%	72.80%	88.14%		88.14%	84.13%	72.80%	88.14%
	Neurons	**BER**	10^{-6}	76.81%	76.14%	71.21%	80.57%	**Regs**	87.08%	81.31%	71.75%	81.62%
			10^{-5}	73.99%	75.96%	70.97%	80.37%		-	-	-	-
	Weights		10^{-6}	65.79%	70.83%	61.11%	74.86%	**FUs**	80.28%	77.10%	65.94%	81.95%
			10^{-5}	30.66%	41.26%	30.64%	43.92%		67.27%	50.35%	71.70%	42.53%
Lenet5	**Golden accuracy**			98.00%	98.00%	98.25%	98.00%		98.00%	98.00%	98.25%	98.00%
	Neurons	**BER**	10^{-6}	83.60%	95.65%	93.19%	94.80%	**Regs**	98.00%	97.10%	97.23%	98.00%
			10^{-5}	75.12%	93.96%	88.10%	93.55%		-	-	-	-
	Weights		10^{-6}	95.37%	97.30%	94.70%	97.40%	**FUs**	93.55%	92.81%	87.09%	95.96%
			10^{-5}	75.54%	95.01%	76.35%	93.50%		86.20%	91.63%	90.51%	91.42%

The results show that APP-level FI overestimates the beneficial impact of SW hardening techniques in terms of both reliability improvement and accuracy degradation. APP FIs tend to understimate the percentage of Critical SDCs and neglect that of DUEs. More in general, moving to a hardware-aware FI technique (hence, intrinsically more accurate) changes completely the ranking of the different hardening techniques: as an example, APP FI identifies Adaptive Clipper as the best solution in terms of reliability enhancement for Mobilenet V2, while for ISA FI the best solution is Swap ReLU6.

In future works, we plan to extend our analysis by exploring further hardening techniques, different underlying HW architectures, and more DNN models.

REFERENCES

[1] A. Libri *et al.*, "paella: Edge ai-based real-time malware detection in data centers," *IEEE Internet Things J.*, vol. 7, no. 10, 2020.

[2] Nvidia Corporation, "Nvidia drive for automotive," https://developer.nvidia.com/drive, 2024, accessed: 2024-08-09.

[3] Advance Micro Devices, "Amd instinct," https://www.amd.com/en/products/accelerators/instinct/mi300.html, 2024, accessed: 2024-08-09.

[4] H. D. Dixit *et al.*, "Silent data corruptions at scale," *arXiv preprint arXiv:2102.11245*, 2021.

[5] P. H. Hochschild *et al.*, "Cores that don't count," in *Proc. of the Workshop on Hot Topics in Operating Systems*, 2021.

[6] A. Singh *et al.*, "Silent data errors: Sources, detection, and modeling," in *IEEE 41st VLSI Test Symp. (VTS'23)*, 2023.

[7] N. Cavagnero *et al.*, "Transient-fault-aware design and training to enhance dnns reliability with zero-overhead," in *IEEE 28th Int. Symp. on On-Line Testing and Robust System Design (IOLTS)*, 2022.

[8] M. Correia *et al.*, "Practical hardening of crash-tolerant systems," in *USENIX Ann. Tech. Conf. (USENIX ATC'12)*, 2012.

[9] W. W. Peterson *et al.*, *Error-correcting codes*. MIT press, 1972.

[10] V. Stefanidis *et al.*, "Algorithm based fault tolerance: Review and experimental study," in *International Conference of Numerical Analysis and Applied Mathematics*. Citeseer, 2004.

[11] Z. Chen *et al.*, "A low-cost fault corrector for deep neural networks through range restriction," in *51st Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2021.

[12] G. Esposito *et al.*, "Enhancing the reliability of split computing deep neural networks," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2024.

[13] E. Ozen *et al.*, "Boosting bit-error resilience of dnn accelerators through median feature selection," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, 2020.

[14] M. Abdullah Hanif *et al.*, "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philos. Trans. R. Soc. A*, vol. 378, no. 2164, 2020.

[15] D. Gnad *et al.*, "Reliability and security of ai hardware," in *2024 IEEE European Test Symp. (ETS)*, 2024.

[16] A. Ruospo *et al.*, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *IEEE 22nd Latin American Test Symp. (LATS)*, 2021.

[17] A. Mahmoud *et al.*, "Pytorchfi: A runtime perturbation tool for dnns," in *50th Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks Workshops (DSN-W)*, 2020.

[18] A. Ruospo *et al.*, "Evaluating convolutional neural networks reliability depending on their data representation," in *23rd Euromicro Conf. on Digital System Design (DSD)*, 2020.

[19] E. Malekzadeh *et al.*, "The impact of faults on dnns: A case study," in *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2021.

[20] T. Tsai *et al.*, "Nvbitfi: Dynamic fault injection for gpus," in *51st Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2021.

[21] S. K. S. Hari *et al.*, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *Int. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2017.

[22] J.-D. Guerrero-Balaguera *et al.*, "Evaluating the impact of permanent faults in a gpu running a deep neural network," in *IEEE Int. Test Conf. in Asia (ITC-Asia)*, 2022.

[23] J. D. Guerrero Balaguera *et al.*, "Reliability assessment of neural networks in gpus: A framework for permanent faults injections," in *IEEE 31st Int. Symp. on Industrial Electronics*, 2022.

[24] J. E. R. Condia *et al.*, "A multi-level approach to evaluate the impact of gpu permanent faults on cnn's reliability," in *IEEE International Test Conference (ITC)*, 2022.

[25] G. Papadimitriou *et al.*, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021.

[26] S. Tselonis *et al.*, "Gufi: A framework for gpus reliability assessment," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2016.

[27] J. E. R. Condia *et al.*, "An effective method to identify microarchitectural vulnerabilities in gpus," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 2, pp. 129–141, 2022.

[28] A. Ruospo *et al.*, "Assessing convolutional neural networks reliability through statistical fault injections," in *Design, Automation & Test in Europe Conf. & Exhibit. (DATE)*, 2023.

[29] J.-D. Guerrero-Balaguera *et al.*, "Evaluating the reliability of supervised compression for split computing," in *IEEE 42nd VLSI Test Symp. (VTS)*, 2024.