One Flip Away from Chaos: Unraveling Single Points of Failure in Quantized DNNs

Cheng Gongye and Yunsi Fei
Department of Electrical & Computer Engineering
Northeastern University, Boston, MA
Email: {gongye.c, y.fei}@northeastern.edu

2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) | 979-8-3503-7394-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/HOST55342.2024.10545351

***Abstract*—Deep Neural Networks (DNNs) have become integral to security-sensitive and mission-critical tasks due to their remarkable performance. However, their deployment faces various security risks, including integrity corruption by fault attacks that disrupt computations or tamper with parameters. While past studies have primarily focused on the vulnerabilities of DNN weights to these attacks, the wider implications of single-bit flips (SBFs) on other parts of DNN implementations have not been investigated. In this research, we target a comprehensive and holistic analysis of the robustness of quantized DNN models against SBFs.**

Utilizing the AMD-Xilinx DPU, an advanced FPGA DNN accelerator, we delve into the tangible repercussions of SBFs on a DNN hardware implementation. Our results reveal that an SBF in about 25% of the bits in an AMD-Xilinx DPU DNN can lead to severe consequences, ranging from application execution failures and system lock-ups to notable inference accuracy losses. Through binary comparison, we pinpoint single points of failure (SPOFs) on the AMD-Xilinx DPU DNN models. On the CPU side, we evaluate the PyTorch TorchScript model format, designed for production deployment on servers, and the results show that SBFs have comparable detrimental effects on DNN software implementations, underscoring the generality of this problem.

Our analysis is based on an effective framework that runs the bit-flipped quantized DNN model on real deployment platforms, hardware accelerators or CPUs, to monitor the consequences of SBFs, allowing for broad assessment across diverse models and datasets. Contrary to the prevailing belief that quantized DNNs are resilient to bit-flips, our systematic analysis offers new insights and finds SPOFs, showing that quantized DNN models are actually very vulnerable to fault attacks. Our work stresses the pressing need for protection strategies for robust DNN inferences in critical applications.

***Index Terms*—Fault tolerance, Deep learning, AI accelerators**

I. Introduction

Deep Neural Networks (DNNs) have emerged as potent tools, showcasing unrivaled performance across diverse fields such as computer vision and natural language processing. As we are witnessing proliferated adoption of DNNs in critical and security-sensitive applications such as autonomous vehicles and robotics, the security of DNN models in these applications, including integrity, confidentiality, and availability, is of paramount importance.

Fault attacks pose serious threats to the integrity of DNN models. The target applications would function incorrectly due to disrupted computations or faults deliberately imposed on model parameters. They can be physically executed via techniques such as Electromagnetic (EM) pulsing [1], laser beaming [2], and clock glitching [3], or enacted through software methods including RowHammer [4], FPGAHammer [5], and GPU overdrive [6]. A variety of platforms are susceptible to fault injections, including CPUs, GPUs, FPGAs, and ASICs.

There exists some prior work that investigates the vulnerability of DNN models with floating-point parameters (weights and biases) to fault attacks [7]. Such models are sensitive to faults: a Single-Bit Flip (SBF) in the Most Significant Bit (MSB) of the exponent in nearly half the weights can severely degrade a DNN model's performance [7]. Quantized DNN models, not only achieve efficiency, but also are widely perceived as more resilient to bit-flip attacks on the weights [7], [8]. Some work has suggested using quantized DNNs as a defensive mechanism against fault attacks [7]–[10]. Other work leverages the resiliency of model weights to faults and intentionally lowers the voltage of a hardware DNN accelerator, inducing random but unharmful errors in the memory/weights, to achieve power savings [11] or to combat adversarial example attacks [12].

Although weights indeed constitute a large portion of a DNN model, other parts of the model implementation, including the code delineating the DNN's structure and operations, the data flow, and activation functions, are equally vital. The existing fault analysis of quantized models is incomplete as it leaves out many critical components of the DNN model.

Our study aims to fill this research void. We examine the ramifications of SBFs in real-world applications. We strategically select two representative applications. On the hardware accelerator side, we use AMD-Xilinx's DPU, a state-of-the-art production-ready FPGA-based DNN accelerator, as our testbed for exhaustive fault analysis. On the CPU side, we perform our fault analysis to another widely-used, production-stage quantized DNN model format—PyTorch TorchScript serialization. Throughout this work, we adhere to each platform's default configurations as specified in official guidelines. We simulate the fault injection process, i.e., manually flipping bits within the target DNN model, and then run the bit-flipped model on actual hardware platforms. This approach captures the realistic impact of a successful bit-flipping attack.

As the consequences of SBFs on a model vary significantly depending on where the SBF falls, we classify the consequences

This work was supported in part by the U.S. National Science Foundation through the following grants: CNS-2212010, CNS-1916762, and SaTC-1929300.

and pay close attention to parts of the model which are Single Point of Failure (SPOF), i.e., exemplifying acute consequences of SBFs. Identification of SPOFs provides useful guidelines for effective and efficient defense mechanisms and mitigation strategies against SBFs.

This work makes the following contributions:

- We discover that quantized DNN models are not immune to SBFs. Specifically, a flip in about one-fifth of the bits in an AMD-Xilinx DPU DNN model, and two-fifths of the bits in a TorchScript DNN model, respectively, can cause severe consequences, including application crashes, system lock-ups, or substantial model accuracy degradation. This research complements the prior work, broadening the evaluation scope beyond the weights of DNN models.
- To our best knowledge, this is the first work that identifies scaling factors of quantized DNN models as SPOFs, with an SBF degrading the accuracy of the model to random guesses. It is generally applicable to almost *any quantized DNN models*, agnostic of the platform.
- We create a framework that is able to accurately capture the effects of SBFs on quantized DNN models, in both hardware accelerators or software implementations. Additionally, we evaluate SBFs on the scaling factors across different DNN models and datasets, providing comprehensive explanations for the fault effects based on the information bottleneck (IB) theory [13].
- We propose a set of mitigation strategies against bit-flip attacks in light of our holistic fault analysis. Our findings suggest that quantized DNN models are as susceptible, if not more so, as other floating-point models, demanding systematic protection schemes to safeguard DNN model inference.

II. Background

This section provides background materials, including fault attacks on DNN models, the two platforms for edge devices and server CPUs respectively, the quantization of DNN models and their different implementations, and related work.

A. Fault Attacks on DNNs

Fault attacks present a formidable challenge to the integrity of DNNs by deliberately disrupting computation processes or tampering with model parameters. Typical fault models include single-bit flips, an effective and stealthy threat to DNN inference.

The RowHammer attack stands out as a frequently used fault injection method for inducing SBFs [7]–[10]. It exploits inherent vulnerabilities in Dynamic Random Access Memories (DRAMs) to precisely flip a targeted bit, which does not require the kernel-level privilege. Many commercial DDR4 memory modules from leading vendors are found to be susceptible to new RowHammer attacks that can effectively bypass the defenses [14]. RowHammer attacks have been successfully applied to DNN models and reduce their inference accuracy to the level of random guessing by strategically flipping selected weight bits [15].

There are various other fault injection methods, including electromagnetic/laser fault injection, clock glitches, and voltage fluctuations. Some can induce SBFs while others introduce random faults to computations or data in memory.

B. AMD-Xilinx DPU

For edge devices, we choose to conduct a comprehensive fault analysis on a representative DNN accelerator, AMD-Xilinx DPU. DPU is one of the most sophisticated state-of-the-art FPGA-based DNN accelerators, with exceptional energy efficiency. It has become a popular platform for real-world applications, including Subaru's production vehicles [16], [17] and Baidu's Apollo autonomous driving system [18].

To ease model deployment on FPGAs by software developers and data scientists, modern FPGA-based accelerators offer comprehensive toolchains, such as Vitis-AI [19]. These toolchains cover the entire workflow, from training a DNN model to its execution on FPGA accelerators. A user can still train a DNN model with floating-point numbers using TensorFlow [20]. The Vitis-AI framework then quantizes the model into 8-bit integers with minimal accuracy loss [21]. The model is then compiled into a binary file executable, typically in the format of executable and linking format (ELF), directly on the accelerator, eliminating the need for bitstream regeneration.

C. PyTorch and TorchScript

For the server CPU side, we select the TorchScript model as the target to perform fault analysis. PyTorch has emerged as a leading framework for training DNN models, commanding over 60% of the market share [22] due to its high performance and flexibility. While the model generated post-training using PyTorch offers many advantages, it is not inherently optimized for rapid inference and deployment and typically relies on Python for execution. To address these challenges and cater to "high-performance production deployment", PyTorch introduced TorchScript in 2018 [23].

TorchScript serialization is a model format that not only facilitates the storage of streamlined quantized models but also offers the flexibility of deployment on systems without using Python, utilizing solely C++ for instance. TorchScript has transitioned to a stable production stage, meaning that the models being deployed are fully fledged. In this study, we leverage the TorchScript model to exemplify the universality of the vulnerabilities in DNN models we have identified.

D. Quantization of DNN Models

Quantization of DNNs significantly optimizes the resources during inference, achieved by substituting original 32-bit floating-point parameters with lower-bit formats, such as 8-bit integers. This method reduces memory usage by nearly four times and lowers the computational cost of multiplication by a factor of sixteen [24], resulting in faster inference by a factor of two to four [21]. Another advantage of quantization is it can be applied post-training, with negligible accuracy loss [21]. Moreover, quantized weights and biases are more resistant to SBFs than floating-point parameters, making them suitable

for security-sensitive applications [7]. As such, quantization is the preferred choice for DNN model inference in edge devices and server CPU deployment. For instance, AMD-Xilinx DPU [25], TensorFlow-Lite [26], and Apache TVM VTA [27] all incorporate 8-bit quantization in their default configuration. The DPU and TVM are specifically designed for quantized models and do not provide support for full-precision DNN models.

E. Quantization in Hardware Accelerator vs. CPUs

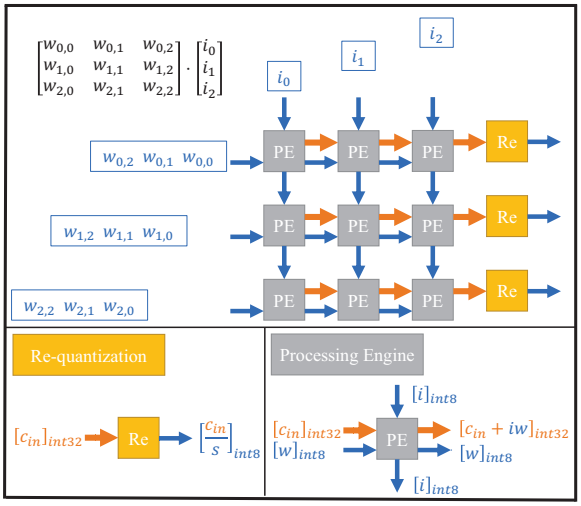

Fig. 1: Illustration of an 8-bit quantized systolic array

Next, we examine the process of quantization on hardware accelerators. Then we highlight the key similarities and subtle differences in its implementation on CPUs. In Fig. 1, we depict a simplified example of how a generic systolic array accelerates the multiplication of a 3x3 weight matrix, $\boldsymbol{W}$, with a size 3 input vector, $\boldsymbol{I}$. As seen in the figure, the core component is an array of Processing Engines (PEs), followed by a vector of re-quantization modules. Each PE takes an input scalar and a weight scalar, multiplies them, and adds the product onto the accumulation input. This operation is Multiply-Accumulate (MAC). Instead of running 32-bit floating-point MAC operations on each PE, quantization facilitates 8-bit integer operations on them, leading to much simpler, smaller, and faster hardware logic of PEs with much lower power consumption [24].

To quantize the floating-point input vector $\boldsymbol{I}$ and the weight matrix $\boldsymbol{W}$ into integers, we need to select two scaling factors s_i and s_w, respectively, and multiply them with the model inputs and weights:

$$\boldsymbol{I_{int}} = \lfloor \boldsymbol{I} \cdot s_i \rceil, \boldsymbol{W_{int}} = \lfloor \boldsymbol{W} \cdot s_w \rceil \tag{1}$$

Here s_i and s_w are carefully selected to minimize the overall errors caused by rounding and clipping [24]. Typically they are integers in high-performance DNN accelerators [24], [27]. The weights undergo offline quantization (multiplying the original weight with the scaling factor), while the inputs are generally subject to online quantization by the inference application. Such quantized weights and inputs are fed to the PEs to compute the following result:

$$\boldsymbol{W_{int} I_{int}} \approx s_w s_i \cdot \boldsymbol{WI} \tag{2}$$

Note that the current output by the PE is $s_w s_i$ times the original value. Moreover, as shown in Fig. 1, the bit-width has grown to 32 due to the MAC operation. To restore the value and prepare an 8-bit input for the next layer, we must add a re-quantization step by dividing the PE output with a scaling factor s that is tailored for the feature map. Generally, *in quantized models, the scaling factors and the re-quantization step are integral parts.* As a cautionary note, there are quantized models that can work without scaling factors, e.g., some binary quantized DNN models whose bit-width is one, which are designed for highly resource-constrained edge devices.

In the CPU environment, the quantization process exhibits similarities to that of hardware accelerators. Both the input and weight are quantized to integer values, and the multiplication operations are accelerated using SIMD (Single Instruction, Multiple Data) instructions, such as SSE (Streaming SIMD Extensions) and AVX (Advanced Vector Extensions) on Intel x86 CPUs. A re-quantization step is also essential following multiplication.

A notable difference arises in the type of scaling factors used. CPUs often employ single-precision floating-point (FP32) scaling factors, as opposed to INT8 scaling factors. Since SSE does not natively support division by INT8 and requires casting to INT32, division by an FP32 scaling factor incurs only a minor performance penalty. More importantly, the use of FP32 scaling factors simplifies the achievement of higher quantization accuracy. It is crucial to note that the data type of the scaling factors does not impact the focus of our fault analysis. Our results will demonstrate that an MSB flip, in either case, can substantially compromise the accuracy of the DNN models.

F. Related Work

To our best knowledge, this research represents the first attempt to perform a comprehensive fault analysis on quantized DNN models, beyond weight and biases. Also, it is the first to reveal that an SBF on the scaling factors of quantized DNN models can dramatically reduce the accuracy, effectively turning it into random guesses. Previous work by Rakin et al. [8] showed that while the weights of quantized models are more resistant to bit-flips than floating-point models, they can still trigger a similar degradation in DNN model accuracy to random guesses when a small number of bits are strategically flipped. Fasfous et al. [28] investigated fault injections in the PEs of DNN accelerators and hinted at training to limit the range of scaling factors. However, they did not explore the effect of SBFs on scaling factors.

III. Hoslitic Fault Analysis of Quantized DNN Models

This section describes our comprehensive fault analysis of quantized DNN models, under the assumption of a single-bit flip.

A. Methodology

To observe the consequences of bit-flips on different parts of a quantized DNN model, we look into the saved model file and modify it, one bit at a time, and run the resulting model on actual hardware. This process ensures that the fault effects are real, reflecting the outcomes that an adversary could trigger by inducing SBFs in a victim's DNN model.

1) Limitations and Generalization: The methodology has an inherent limitation: the results are tied to the specific implementation being examined. While it is infeasible to encompass the vast varieties of existing DNN implementations, we strategically choose two implementations to enhance the generalizability of our findings.

Our primary target is AMD-Xilinx DPU, one of the most advanced DNN processing units available, representing hardware implementations of DNN. DPU provides a robust environment for a victim DNN model, with complex hardware architecture and encrypted IP. Utilizing a state-of-the-art implementation ensures that the observed fault effects are not artifacts of an underdeveloped or rudimentary system. Our second target is the widely-used TorchScript DNN model, representing general software implementations on CPUs or edge MCUs.

With two different implementations, we focus on identifying vulnerabilities that are agnostic to the specific hardware or DNN framework, thus contributing to a more general understanding of the fault behavior.

2) Practical Challenges: Conducting exhaustive fault analysis is computationally expensive and time-consuming, particularly for large and complex models. We start our study with a simple yet representative task: classification of the handwritten MNIST dataset [29]. The selected DNN architecture, illustrated in Fig. 2, comprises two convolutional layers (Conv 1 and Conv 2), each followed by a max-pooling layer. These layers are followed by a dense layer to generate the output logits.

Another challenge for experiments with bit-flips is the frequent application execution failures and device lock-ups of the device-under-test (DUT), which render the DUT unresponsive, necessitating a *manual* hard reset. To address this problem, we devise a monitoring program on a separate single-board computer, enabling us to capture faulty results and *automatically* hard reset the DUT using a MOSFET relay when needed. This approach obviates the need to manually press the reset button for ten seconds every time the system locks up. Given the high frequency of system lock-ups, the experiment would not be feasible without this measure.

B. Experiment Setup

The architecture of the AMD-Xilinx DPU is tightly integrated with the AMD-Xilinx System-on-Chips (SoCs) [30]. Each SoC includes ARM cores, referred to as the Processing System (PS), and Field-Programmable Gate Arrays (FPGAs), designated as Programming Logic (PL). These components are seamlessly interconnected via an on-chip AXI bus, enabling a cohesive hardware-software co-design paradigm.

Within this architecture, the Domain-Specific Accelerator (DSA) resides on the PL side and is configurable via the AXI

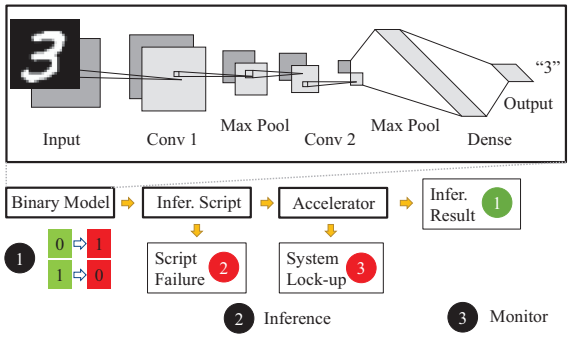

Fig. 2: Overview of the fault analysis methodology.

interface. The PS side not only runs a Linux-based operating system but also orchestrates the application execution with scripts. It delegates computationally-intensive tasks (e.g., matrix MACs) to the PL-side accelerator and handles the results thereof. The PS-side application loads the model into the DRAM, enabling the DSA to access it via direct memory access.

1) Factors Influencing Fault Resilience: The fault resilience of an AMD-Xilinx DPU-based DNN model is contingent upon three salient factors:

1) The robustness of the PS-side script when confronted with corrupted model files.
2) The inherent resilience of the DNN model parameters (weights and biases) in the DRAM to bit-flips.
3) The DSA's capability to gracefully handle corrupted instructions and configurations.

Complementing the prior work that predominantly focuses only on the second factor, our work examines the full landscape of DNN models for fault resilience.

2) Experiment Workflow: As depicted in Fig. 2, our experiment proceeds in three stages:

- **Stage** 1: Modification of the target binary model file by flipping a single selected bit.
- **Stage** 2: Activation of the inference script, which loads the modified model into the DSA.
- **Stage** 3: Monitoring of the execution outcome on a separate platform.

We categorize the observed results of model execution under an SBF into three scenarios:

1) Successful execution with potential degradation in inference accuracy. The model is executed multiple times to compute the average classification accuracy (1).
2) Script failure without system lock-up, where the application aborts and the next bit-flip is subsequently tested (2).
3) Extreme cases with system lock-up, e.g., device freezes or is unresponsive, necessitating a system reboot or hard reset (3).

3) Setup Details: We employ the Ultra-96 V2 as our DUT [31], which is a widely recognized reliable SoC evaluation board compatible with DPU. The board runs the PYNQ

Linux v2.5 [32], specifically tailored for DPU applications by AMD-Xilinx. Note that we leave the DPU bitstream and all default configurations unaltered throughout the experimentation process. Vitis-AI Docker version 1.2.82 is used. The quantized CNN model trained on MNIST attains a test accuracy of 93.12%, comparable to the floating-point version. The compiled binary model has a size of 5.8 KBytes, i.e., approximately 47 K bits for receiving SBFs. Each bit-flip is tested against the testing dataset of $10,000$ images.

C. Experimental Result Overview

For Outcome (1), we quantify the accuracy degradation as the relative accuracy loss compared to the original accuracy, shown in the formula below:

$$D = 1 - \frac{a}{A} \tag{3}$$

where A represents the testing accuracy of the original DNN model and a is the observed accuracy of the faulty model.

Considering the MNIST dataset with its ten classes, the worst possible observed accuracy a would be $\frac{1}{10}$, equivalent to randomly guessing the outcome. Consequently, the largest possible degradation would be $D = 1 - \frac{0.1}{0.9312} = 0.8926$; while the smallest possible degradation is 0, i.e., unaffected accuracy. As illustrated in Fig. 3, this unaffected accuracy is the most common result type, accounting for approximately 64% of the cases. This number indicates a certain level of fault resilience of the model implementation. We categorize an accuracy degradation as minor when D is below 0.2, and severe otherwise. 4.5% of bit-flips account for severe accuracy degradation. The result also shows that for the other two serious outcomes (more like denial-of-service), each is contributed by about 8% of the bit-flips.

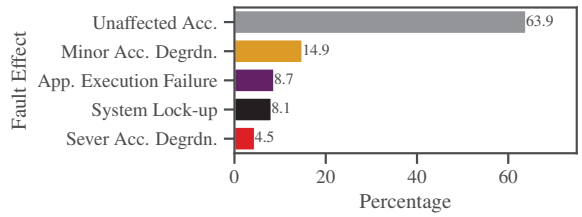

Fig. 3: Percentage distribution of SBF fault effects on a DPU model

The overall conclusion of fault analysis for the AMD-Xilinx DPU is: *approximately* 20% *of the bits in a DNN model act as Single Point of Failures (SPOFs), leading to severe performance degradation, application execution failures, or system lock-ups.* To corroborate these findings and ensure that they are not isolated incidents, we conduct the experiment on a larger DNN model of size 10.1 KB. We observe a similar proportion of these failure points even in larger models. Only the code and weight/bias regions undergo considerable expansion as the model size increases. The ratio of SPOFs within these regions remains largely unaltered. This uniformity across different model sizes substantiates the importance of our findings and their broader applicability.

D. Experiment Result Discussion

We next dive into analyzing the observed results. The effects of flipping each bit of the model, depending on different model regions, are depicted in Fig. 4. We have organized the bits into four-byte blocks. For each plot, the X-axis represents the block index of the binary file, while the Y-axis signifies the bit index within each block. Thus, each dot on the plot corresponds to the result of one bit-flipping experiment. We use different colors, as defined in the legend, to represent the aforementioned five types of fault effects. For inference accuracy degradation (ranging from 0 to 0.89), the size of each dot corresponds to the level of degradation. For visual clarity, dots for both application execution failures and system lock-ups are also at the size of the highest possible degradation.

The binary DNN model is an ARM executable and linking format (ELF) file [33]. Our analysis reveals that it functions as a shared object file and contains multiple regions, which are all closed-source and proprietary. While we lack detailed understanding, such as how to decode its instruction set architecture (ISA) or the precise functions of each region, we discover that the region names were not obfuscated. Each region is graphically represented in a separate subplot, as illustrated in Fig. 4. In addition to the region title, we also report its size and the percentage of bits that we classify as SPOFs.

Next, we analyze the effects of bit-flips in various regions one by one, alluding to the potential causes.

a) ***String Table****:* These regions are dedicated to storing ASCII strings. For String Table I, the majority of the fault effects are *unaffected accuracy* or *application execution failures* (7.72%). Notably, two strings found in the string table I, specifically `conv2d_1_convolution` and `output_logits_MatMul`, are responsible for script failures. These strings serve as identifiers for the input and output layers, respectively, as designated by the Vitis-AI framework. Should the script fail to locate these expected names, it terminates prematurely. The String Table II is found towards the end of the binary file, comprising region names. The majority of SBFs in this region do not result in script failures, but instead more serious system lock-ups. The DPU driver fails to process the corrupted content when certain hard-coded region names are compromised, rendering it unresponsive to inputs.

b) ***Configuration****:* All three types of severe fault effects show up for this region. Decoding this region poses a challenge due to the proprietary and closed-source nature of the DPU. Judging from its name, it likely stores the configuration values for the DSA's controlling registers on the PL side. Script failures and system lock-ups in this region are probably triggered by invalid configurations. We managed to recover that some bytes are for the sizes of tensors. Interestingly, flipping these tensor sizes has no effect on the performance of the DNN models, suggesting they are redundant.

c) ***Weight and Bias****:* This region serves as a repository for the weights and biases of all DNN model layers. Echoing previous research findings, the majority of bit-flips (on weights) barely impact the classification accuracy. However, the bias of

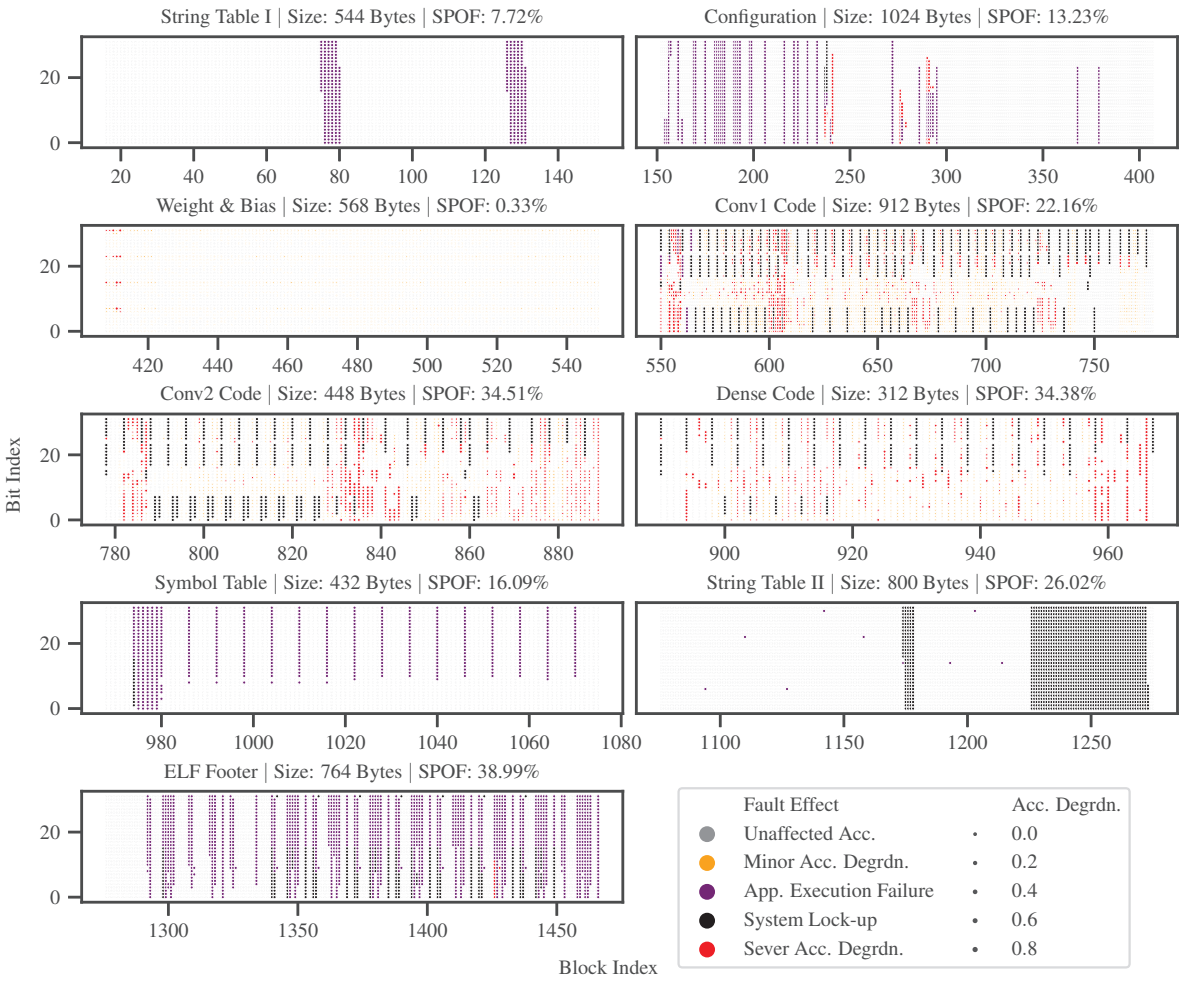

Fig. 4: Visualization of the individual SBF effects on an AMD-Xilinx DPU binary DNN model

the output layer behaves differently. Specifically, flipping the MSB and the second MSB of the ten biases of the ten output nodes dramatically reduces the classification accuracy, with the worst accuracy sharply dropping to 46.28%. This region is free of other two types of severe results: application failure and system lock-up.

d) ***Code****:* There are three code regions, two convolutional and one dense, and every one of them exhibits a lot of SPOFs. Note the max-pooling layers have been integrated into the convolution layers, and the two dense layers have merged. The DPU ISA is proprietary and we cannot disassemble the code, but we can still glean some insights using coarse-grained information. The first observation is that the length of the code region correlates directly with the number of arithmetic computations in each layer. Given that the accelerator functions by segmenting the workload, each instruction group (following the convention of four bytes per instruction) handles a small segment of the task. This segmented approach effectively enables the code to operate in a manner akin to loop unrolling. Second, the error types in the code region follow a highly regular, periodic pattern, confirming the loop-unrolling fashion. Starting from the bottom and moving upwards within the code regions, it is observed that bit-flips in the first, third, and fourth bytes of certain instructions/blocks are highly prone to inducing system lock-ups (shown as black dots). These instructions are most likely associated with load and save operations; consequently, tampering with their critical fields has the potential to corrupt memory content severely, leading to system-wide failures. In contrast, there are other instructions where bit-flips only result in severe model accuracy degradation (shown as red dots). We will elaborate on our efforts to identify these instructions and discuss why they contribute to accuracy degradation in Section III-F (we find them to have scaling factors hard-coded in instructions, in the immediate field).

e) ***Symbol Table and ELF Footer****:* Following the ELF specification, these regions contain vital information for iden-

tifying and repositioning a program's symbolic references and definitions. As such, it has a high likelihood of causing application execution failures.

Considering the different regions would manifest different fault effects, various strategies can be developed accordingly to meet the attacker's goal. If the intention is to undermine the availability of the DNN application without destabilizing the system, they could focus on the ELF footer, where nearly 40% of the bits can lead to application execution failure. If the objective is to immobilize the system, they could aim for the end of the String Table II, where almost all bits could achieve this outcome. Alternatively, if the aim is to deteriorate the accuracy, numerous bits in the code region could precipitate varying degrees of accuracy degradation.

Given the outcomes of our analysis, it is crucial to recognize that the design of these regions in the AMD-Xilinx DPU DNN model does not seem special. As one of the most sophisticated DNN accelerators currently on the market, its architectural design and functionality align with the industry's state-of-the-art standards. The vulnerabilities to bit-flips that we identified tend to be common and could presumably be observed in other advanced systems.

Furthermore, it is conceivable that other systems, particularly those less complex ones, could exhibit similar or even heightened susceptibility to such faults. Hence, while our analysis is tailored to the AMD-Xilinx DPU, The conclusions and insights derived could offer valuable contributions to understanding and enhancing a broader range of DNN accelerators.

E. Comparison with Fault Analysis of PyTorch TorchScript Model

Our fault analysis on the AMD-Xilinx DPU model reveals a critical insight: the code region is significantly more susceptible to SBFs than the data region.

It is plausible that such fault vulnerabilities are due to the particular DPU implementation, e.g., the composition of each region. To make our analysis general to models and not specific to implementations, we extend our study onto PyTorch TorchScript, another widely adopted format for production-stage DNN models.

a) Experiment Setup and Methodology: To ensure a fair and direct comparison between the AMD-Xilinx DPU and TorchScript models, our experiment setup and methodology closely mirror those outlined for the DPU in Sec. III-B. In this section, we elaborate on the key differences between these two sets of experiments.

The DNN models have identical structures, hyperparameters, and parameters, as those used in experiments on AMD-Xilinx DPU. We employ the latest stable release PyTorch 2.0.1 and Python 3.11. The model weights are quantized to 8-bit integers and inference is accelerated using the Intel AVX extension. The experiments are conducted on an Intel i5-10600K CPU equipped with 64 GB of DDR4 2667 MHz memory and running Ubuntu 18.04.6. We also use the default quantization configurations and code provided by the official PyTorch website. Notably, despite the same DNN model, the file size of the TorchScript model is substantially larger—26.4 KB—compared to the DPU model, which is 5.8 KB.

A key distinction between the DPU and TorchScript models lies in their handling of input quantization: while the TorchScript model autonomously manages input quantization, the DPU model depends on the application script to supply quantized inputs. In PyTorch's default configuration, input tensor quantization involves scaling the floating-point values with an FP32 scaling factor, offsetting them with an INT64 zero-point value, and subsequently rounding them to INT8 values. Furthermore, TorchScript stores scaling factors and zero-point values in the data region as if they were data, whereas the DPU model does not. The implications of these differences will be elaborated upon in the section discussing experimental results.

b) Experiment Results and Discussion: As depicted in Fig. 5, the TorchScript model exhibits a higher percentage of SPOFs (approximately 41% and mostly execution failure) than the DPU model, indicating the model is more vulnerable. The percentages of system lock-ups and accuracy degradation are both low.

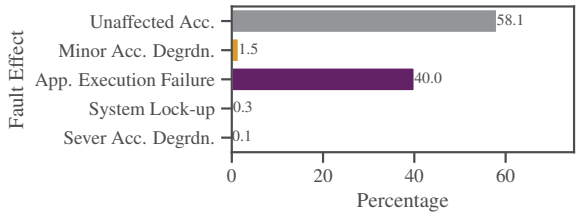

Fig. 5: Percentage distribution of fault effects on a TorchScript model

The TorchScript format enjoys widespread adoption for deploying high-performance model inference in production, leveraging PyTorch's popularity. A TorchScript model primarily comprises two components: the code region and the data region. The portable and self-contained TorchScript code outlines the model's structure and inference process. It does not have low-level components like string tables or symbol tables because TorchScript is a high-level language. The data region stores tensors, which include weights, biases, and scaling factors. Similarly, a DPU model also has code region and data region. Both types of models maintain a code region because the code functions as an *integral component* of any DNN architecture, delineating essential hyperparameters such as activation functions, padding, stride, and tensor data flow pathways. Consequently, any fault analysis that excludes the code region would be intrinsically incomplete. Our fault analysis result supports this argument, which shows that the code region in DNN models is especially vulnerable to faults. Due to space constraints, Fig. 6 provides a selected visualization of fault effects on the TorchScript model, concentrating on two representative parts: the model header (part of the code region) and the tensor (part of the data region).

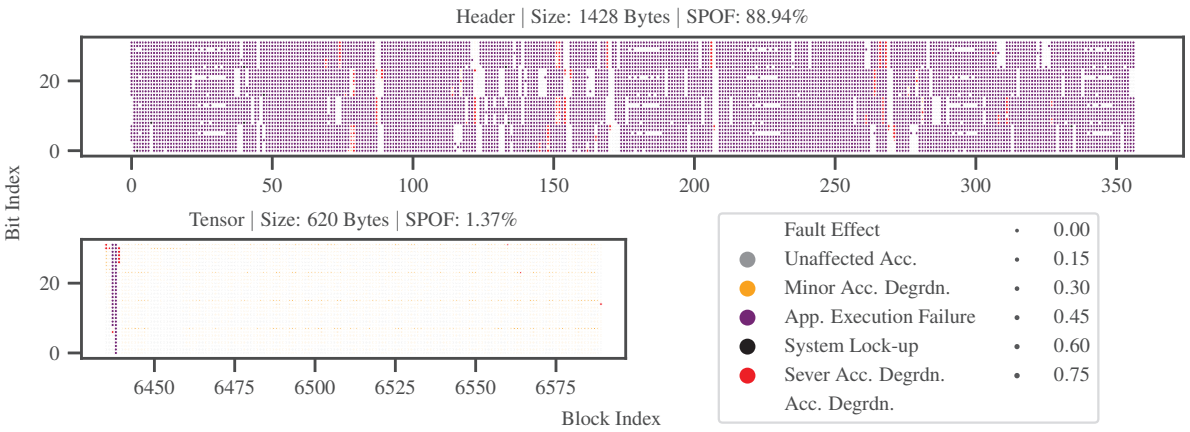

Fig. 6: Visualization of the individual SBF effects on a TorchScript DNN model

Comparing the two model implementations, the TorchScript model's code region appears to be considerably more vulnerable to faults than that of the DPU model. As illustrated in Fig. 6, in the Header region, nearly 90% of the SBFs lead to application execution failures. Similar rates of SPOFs are observed across all other code regions. The overall percentage of SPOFs in the model is offset by the debug regions, which are of the same size as the code regions but are not involved in inference. The higher vulnerability of TorchScript can be attributed to its nature as a high-level language, unlike the DPU's concise binary code. This makes TorchScript less tolerant of SBFs, as it relies on an interpretation engine, leading to earlier failure modes. This observation is corroborated by Fig. 5, which indicates fewer instances of severe accuracy degradation or system lock-ups for the TorchScript model.

Both DPU and TorchScript data regions consist of densely packed bytes, sized at 568 B and 620 B, respectively. The slightly larger size for TorchScript arises from the inclusion of scaling factors and zero points. As illustrated in Fig. 6, the Tensor region features several 4-byte blocks arranged from left to right. The initial two blocks denote the FP32 scaling factors of the first and second layers of the DNN model. They are followed by two 4-byte blocks representing the INT64 zero point, which is exclusive to the input layer. The fifth 4-byte block contains the FP32 scaling factor for the input layer itself. Note that each input value undergoes an addition operation with this zero point after being multiplied by the associated scaling factor. The remaining blocks encapsulate the model's INT8 weights and biases.

Our analysis uncovers a relatively higher susceptibility to SPOFs of the Tensor region (compared to the parameter region of DPU models), attributable to these additional quantization parameters. For the input layer, bit flips in the TorchScript model's scaling factors or zero points often result in application execution failures or severe accuracy drops. In contrast, the DPU model relies on pre-quantized inputs and is obviated from these extra elements. For the rest of the layers, bit flips that impact the exponent of the scaling factor are also detrimental to the model's accuracy, a phenomenon evident in the scaling factors of both the first and second layers. We delve deeper into the ramifications of these fault effects in the subsequent section.

F. Scaling Factors: The Ubiquitous Vulnerability

We have demonstrated the susceptibility of quantized DNN models, specifically on AMD-Xilinx DPU and TorchScript platforms, to SBFs. While there are implementation-specific vulnerabilities, we aim to identify SPOFs that may be universally present across all quantized DNN models.

Out of the model parameters, the MSBs of the biases in the output layer could become potential SPOFs. However, their effects are not universally catastrophic. For example, flipping the biases in the TorchScript model did not produce equally severe degradation. We aim to identify bit-flips that can plunge the model's accuracy to that of random guessing. Our analyses suggest that scaling factors may be the SPOFs of universal concern: they are part of the model's data according to PyTorch's design and should thus be present across implementations; yet the DPU model does not seem to store these scaling factors in its data region, leaving us uncertain whether they are among its SPOFs. This raises two critical questions: Are scaling factors embedded in DPU models, and if so, are they SPOFs?

To answer these questions, we trained multiple DNN models with identical architectures and compiled them into ELF models. Using a Python-based binary difference tool developed in-house, we performed a *differential binary analysis* on these ELF binaries. This tool dissects the binary into various sections and compares 4-byte blocks within those sections to identify differences.

After running pairwise differential analyses on a large set of DPU models, we noted both expected discrepancies in the data regions and unexpected variations in the code regions. Models with identical scaling factors revealed identical code

segments, confirming that scaling factors are indeed integrated into the DPU model structure (in the specific code regions, such as CONV1 Code, CONV2 Code, and Dense Code, as shown in Fig. 4). Subsequent cross-referencing with our fault analysis verified that these scaling factors are SPOFs in the DPU models. In the following section, we elaborate on why scaling factors in quantized DNN models stand out as SPOFs.

IV. Generalizability of Single Points of Failure

We have identified the scaling factors of quantized DNN models as the most notable SPOFs, capable of reducing the model's accuracy to mere random guesses. This section analyzes why quantized DNN models are inherently reliant on scaling factors. We further analyze the impact of SBFs on scaling factors of large DNN models.

Scaling factors are judiciously chosen to minimize the model accuracy loss induced by a lower bit precision. To find an optimal set of scaling factors, complex optimization problems must be solved across different layers [24]. As expected, any change in the value of the scaling factor by SBFs significantly affects the accuracy of the model. Fig. 7 illustrates the effect of the MSB SBF on the scaling factor of the re-quantization step. For an unsigned 8-bit integer scaling factor $134|(10000110)$, upon the normal re-quantization, the feature map follows a normal distribution (the left plot), akin to the original floating-point feature map. However, with an SBF on the MSB of the scaling factor, it is changed to $6|(00000110)$, and a majority of the output values are clipped due to overflow or underflow and concentrates on two end-values (shown in the right plot), resulting in the information entropy dropping from 4.9 bits to 2.0 bits, nearly 60% loss. Consequently, the inference accuracy of the faulty model declines significantly, reaching the point of random guesses when using the MNIST dataset.

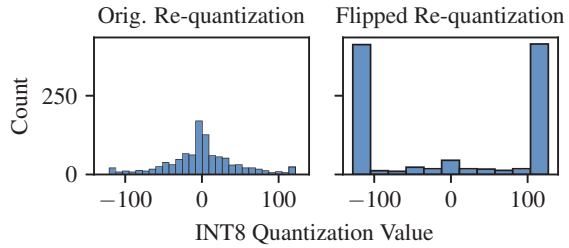

Fig. 7: Effect of SBFs on the scaling factor of re-quantization

We extend such fault analysis to a larger dataset and models. To ensure the results' broad applicability, a generic quantization framework independent of specific hardware is selected [34]. The ImageNet-1K dataset [35], one of the largest publicly available datasets, is chosen with DNN models ResNet-18 and ResNet-50 [36], which achieve top-1 accuracies of 72.2% and 77.3%, respectively, representing state-of-the-art quantized models. The weights in these models are 8-bit integers, and the scaling factors are 8-bit unsigned integers. An SBF experiment targeting the scaling factors used in the re-quantization step is conducted. Fig. 8 shows the results of flipping the MSB of each layer's scaling factor of ResNet-18. Over 50% of SBFs degrade the model's accuracy to the level of random guesses (with the accuracy of $\frac{1}{1000}$ considering the thousand classes in the ImageNet-1K dataset). Two trends are observed: the closer to the input layer, the model is more sensitive to the SBF on its scaling factor; and the odd-number layers are less susceptible to SBFs with higher accuracies. Both phenomena can be explained by the Information Bottleneck (IB) theory [13] and the residual connections structure of ResNet. Similar results are observed in fault analysis of the larger ResNet-50, where 35% of the MSB SBFs degrade the model accuracy to random guess levels.

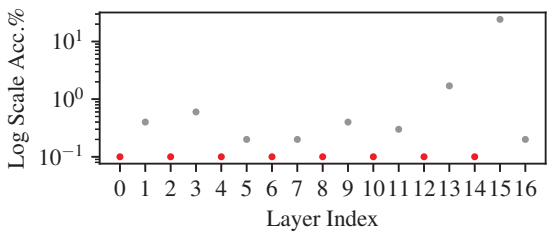

Fig. 8: Logarithm base 10 of the accuracy (in percentage) for the of the scaling factor (MSB SBF) in ResNet-18 evaluated on the ImageNet-1K dataset. Red indicates performance equivalent to random guessing.

The IB theory provides a valuable lens to understand how DNNs process and condense information. It was initially proposed for extracting significant components of an input random variable in relation to the output and has been repurposed for deep learning applications. Each DNN layer is conceptualized as a compression stage, with data flowing through a series of "bottlenecks". This created a Markov chain from input to output. The Data Processing Inequality (DPI) principle ensures each layer contains no more information about the output than its input, maintaining the integrity of information flow through the network [13]. The DPI principle offers a theoretical basis for the empirical observation of increased SBF sensitivity closer to the input layer. The severe accuracy drop induced by SBFs, especially in earlier layers, can be seen as excessive information loss due to a perturbed scaling factor. A distorted scaling factor can lead to information destruction, thereby obstructing the flow of information through the network. The metric used to measure the information content is *Shannon entropy* of the feature maps [13]. As depicted in Fig. 7, a perturbed scaling factor can substantially reduce the information entropy. Specifically, for all the layers of the ResNet-18, a flip in the MSB of the scaling factors leads to an average reduction in information entropy by 97%.

Moreover, the IB theory aids in deciphering the observed resilience of layers with residual connections (odd-number layers). Residual connections serve as parallel information channels, circumventing the information destruction caused

by the SBF. Consequently, the information loss is mitigated, allowing the model to retain reasonable performance despite the perturbation. However, residual connections and SBFs are pitted against each other, and still significant information loss is attributed to the SBFs. This reinforces the importance of scaling factors in the operation of quantized DNN models and highlights the need to devise strategies for safeguarding these factors against bit-flip faults.

V. Discussions

A. Potential Defenses

This paper reveals that, contrary to prior assumptions, quantized DNN models contain numerous SPOFs, therefore, they are not inherently robust to SBFs. Based on this new understanding, we evaluate the potential for adapting existing defenses to safeguard against these SPOFs. A significant challenge is that it is straightforward for attackers to induce an SBF, as a variety of fault injection methods, even those targeting ECC memory and registers during runtime, can accomplish this. We classify and discuss these defenses by category and provide our recommendations.

Defenses Dependent on Quantization: Evidently, defenses that strongly depend on the fault resiliency of quantized DNN models (weights and biases) are not applicable anymore [7]–[10], as their base assumption is no longer valid and their fault analysis is incomplete without considering scaling factors.

Learning-Based Defenses: Learning-based strategies aim to enhance the robustness of DNN models against bit-flips by equalizing the significance of channels and weights [37], or by limiting the range of activations [38]. Unlike bit-flip attacks that target the weight, the scaling factor affects all channels of an entire layer. Furthermore, a scaling factor SBF can decrease, rather than increase, the feature map values. Hence, it is not a straightforward task to extend these approaches to protect against scaling-factor SPOFs.

Checksum-Based Detection: Detection strategies that compute the checksum [39], [40] of model weights could be adapted to shield these SPOFs. However, the area requiring protection is significantly larger than previously assumed. For example, as shown in Fig. 4, weights and biases only represent a small part (around 9%) of a practical quantized model. Extending the checksum to the entire model file may result in substantial overhead. Moreover, these detection-based approaches are susceptible to denial-of-service attacks, as the model needs to be reloaded when a fault is detected, which can be costly. To mitigate the reloading expense, Liu et al. suggest identifying and reloading only the vulnerable weights [41]. However, we have demonstrated that a considerably larger number of elements, i.e. all the SPOFs, require reloading.

We conclude that it is challenging to adapt these defenses to address the discovered SPOFs. For mission-critical and security-sensitive applications, triple redundancy provides a well-known and time-proven solution, despite its significant resource overhead. It can ensure both availability and integrity. If resources cannot support triple redundancy, multi-level defenses are required to counteract both memory and transient faults. For memory faults, ECC memory can be employed. To counter transient faults, ECC could be added at the software level to each 4-byte block of the binary DNN model. Additionally, we need to introduce extra hardware blocks to process the ECCs. Both the driver and the application need rigorous testing against corrupted model files to prevent unforeseen exits and system lock-ups. In summary, there is no universal solution; comprehensive defenses must be considered to counteract these SPOFs.

B. Caveats and Future Work

The binary model of AMD-Xilinx DPU is proprietary and closed-source. Our efforts have identified the scaling factors as SPOFs that can significantly degrade the model accuracy to the level of random guessing. Nonetheless, other SBFs can lead to similar outcomes. However, decoding the semantics of these bits remains a challenge, which leaves us uncertain whether these identified SPOFs are implementation-dependent or universally applicable to all quantized models. Future research could focus on modeling a generic DNN accelerator and its ISA, so as to further explore potential hidden SPOFs.

VI. Conclusion

This study provides an in-depth analysis of the resilience of quantized DNN models in the face of SBFs, specifically for real-world applications with AMD-Xilinx DPU, an advanced commercial FPGA DNN accelerator. Contrary to the common belief, we found that quantized DNN models are not immune to SBFs. A single bit-flip in about one-fifth of the bits in an AMD-Xilinx DPU DNN model could trigger severe consequences such as application crashes, device freezes, or significant accuracy degradation.

Our research has successfully broadened the scope of evaluations beyond the weights and biases of DNN models, contributing to a more comprehensive understanding of SBF vulnerabilities in these models. The differential binary analysis conducted during our research enabled us to reverse engineer the proprietary binary format and identify specific SPOFs not only for the AMD-Xilinx DPU but also for almost all quantized DNN models across diverse platforms. This discovery marks a significant advancement in our understanding of fault vulnerabilities in DNN models.

Furthermore, we developed a framework capable of accurately monitoring the bit-flip effect in quantized DNN model implementations, either in hardware accelerators or in software. This tool, combined with our evaluation of general SPOFs across different DNN models and datasets, allows for an in-depth explanation of the fault effects based on the information bottleneck (IB) theory.

Lastly, our investigation into various mitigation strategies against bit-flip attacks has reaffirmed the need for robust protection measures in DNN model applications. Our research demonstrates that these applications are as susceptible if not more so, to bit-flip attacks than other applications, which underscores the necessity for a systematic and comprehensive protection scheme to secure DNN model inference.

REFERENCES

[1] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2013, pp. 77–88.

[2] J. G. Van Woudenberg, M. F. Witteman, and F. Menarini, "Practical optical fault injection on secure microcontrollers," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2011, pp. 91–99.

[3] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2011, pp. 105–114.

[4] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[5] J. Krautter, D. R. Gnad, and M. B. Tahoori, "FPGAhammer: Remote voltage fault attacks on shared fpgas, suitable for dfa on aes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[6] M. Sabbagh, Y. Fei, and D. Kaeli, "A novel GPU overdrive fault attack," in *ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[7] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks." in *USENIX Security Symposium*, 2019, pp. 497–514.

[8] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1211–1220.

[9] A. S. Rakin, L. Yang, J. Li, F. Yao, C. Chakrabarti, Y. Cao, J.-s. Seo, and D. Fan, "Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy," *arXiv preprint arXiv:2103.13813*, 2021.

[10] L. Liu, Y. Guo, Y. Cheng, Y. Zhang, and J. Yang, "Generating robust DNN with resistance to bit-flip based adversarial weight attack," *IEEE Transactions on Computers*, 2022.

[11] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 267–278, 2016.

[12] S. Majumdar, M. H. Samavatian, K. Barber, and R. Teodorescu, "Using undervolting as an on-device defense against adversarial machine learning attacks," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 158–169.

[13] S. S. Lorenzen, C. Igel, and M. Nielsen, "Information bottleneck: Exact analysis of (quantized) neural networks," *arXiv preprint arXiv:2106.12912*, 2021.

[14] P. Jattke, V. Van Der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 716–734.

[15] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security Symposium*, 2020.

[16] "Automotive Solutions," 2023. [Online]. Available: https://www.amd.com/en/solutions/automotive.html

[17] M. Chiappetta, "Subaru Taps Xilinx For Its New EyeSight Vision-Based Advanced Driver-Assistance System," Aug. 2020. [Online]. Available: https://www.forbes.com/sites/marcochiappetta/2020/08/19/subaru-taps-xilinx-for-its-new-eyesight-vision-based-advanced-driver-assistance-system/?sh=2336702c71f5

[18] W. G. Wong, "Xilinx SoC FPGA Powers Baidu's Apollo Driverless Platform," Jan. 2020. [Online]. Available: https://www.electronicdesign.com/markets/automotive/article/21119589/xilinx-soc-fpga-powers-baidus-apollo-driverless-platform

[19] "Vitis AI," May 2023. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html

[20] "TensorFlow," Apr. 2023. [Online]. Available: https://www.tensorflow.org

[21] "Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT | NVIDIA Technical Blog," Nov. 2022. [Online]. Available: https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt

[22] R. O'Connor, "PyTorch vs TensorFlow in 2023," *News, Tutorials, AI Research*, Apr. 2023. [Online]. Available: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023

[23] "TorchScript for Deployment — PyTorch Tutorials 2.0.1+cu117 documentation," Aug. 2023. [Online]. Available: https://pytorch.org/tutorials/recipes/torchscript_inference.html

[24] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[25] "Introduction • Zynq DPU Product Guide (PG338) • Reader • AMD Adaptive Computing Documentation Portal," May 2023. [Online]. Available: https://docs.xilinx.com/r/3.3-English/pg338-dpu

[26] "TensorFlow Lite | ML for Mobile and Edge Devices," Apr. 2023. [Online]. Available: https://www.tensorflow.org/lite

[27] apache, "tvm-vta," May 2023, [Online; accessed 19. May 2023]. [Online]. Available: https://github.com/apache/tvm-vta

[28] N. Fasfous, L. Frickenstein, M. Neumeier, M. R. Vemparala, A. Frickenstein, E. Valpreda, M. Martina, and W. Stechele, "Mind the scaling factors: resilience analysis of quantized adversarially robust cnns," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 706–711.

[29] "The mnist database of handwritten digits," http://yann.lecun.com/exdb/mnist/.

[30] "Zynq UltraScale+ MPSoC," May 2023. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[31] Digilent. Ultra96-v2 board. https://www.avnet.com/wps/portal/us/products/new-product-introductions/npi/aes-ultra96-v2/.

[32] Xilinx. Pynq. http://www.pynq.io/.

[33] Arm, "ARM ELF File Format," Dec. 2021. [Online]. Available: https://developer.arm.com/documentation/dui0101/a

[34] Q. Jin, J. Ren, R. Zhuang, S. Hanumante, Z. Li, Z. Chen, Y. Wang, K. Yang, and S. Tulyakov, "F8net: Fixed-point 8-bit only multiplication for network quantization," *arXiv preprint arXiv:2202.05239*, 2022.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, 2015.

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

[37] N. Cavagnero, F. Dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Transient-fault-aware design and training to enhance dnns reliability with zero-overhead," in *IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2022, pp. 1–7.

[38] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 1–13.

[39] M. Javaheripi and F. Koushanfar, "Hashtag: Hash signatures for online detection of fault-injection attacks on deep neural networks," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[40] Y. Guo, L. Liu, Y. Cheng, Y. Zhang, and J. Yang, "Modelshield: A generic and portable framework extension for defending bit-flip based adversarial weight attacks," in *IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 559–562.

[41] Q. Liu, J. Yin, W. Wen, C. Yang, and S. Sha, "NeuroPots: Realtime proactive defense against Bit-Flip attacks in neural networks," in *32nd USENIX Security Symposium*, 2023, pp. 6347–6364.