# Derailed: Arbitrarily Controlling DNN Outputs with Targeted Fault Injection Attacks

Jhon Ordoñez and Chengmo Yang
*Department of Electrical & Computer Engineering*
*University of Delaware*
Newark, USA
{jordonez,chengmo}@udel.edu

***Abstract*—Hardware accelerators have been widely deployed to improve the efficiency of DNN execution in terms of performance, power, and time predictability. Yet recent studies have shown that DNN accelerators are vulnerable to fault injection attacks, compromising their integrity and reliability. Classic fault injection attacks are capable of causing a high overall accuracy drop. However, one limitation is that they are difficult to control, as faults affect the computation across random classes. In comparison, this paper presents a controlled fault injection attack, capable of derailing arbitrary inputs to a targeted range of classes. Our observation is that the fully connected (FC) layers greatly impact inference results, whereas the computation in the FC layer is typically performed in order. Leveraging this fact, an adversary can perform a controlled fault injection attack even to a black-box DNN model. Specifically, this attack adopts a two-step search process that first identifies the time window during which the FC layer is computed and then pinpoints the targeted classes. This attack is implemented with clock glitching, and the target DNN accelerator is a DPU implemented in the FPGA. The attack is tested on three popular DNN models, namely, ResNet50, InceptionV1, and MobileNetV2. Results show that up to 93% of inputs are derailed to the attacker-specified classes, demonstrating its effectiveness.**

***Index Terms*—Fault injection, DNN accelerator, Clock glitching**

## I. Introduction

Deep neural networks (DNNs) are widely used in many applications, such as face recognition, object detection, segmentation, and natural language processing. To facilitate the execution of DNNs on resource-constrained devices, various hardware accelerators have been developed, including Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), and Application Specific Integrated Circuits (ASICs). Unfortunately, the advances in DNN deployment also expose them to various security threats [1]. One threat is faults which can disturb the circuit functionality to either produce malicious/meaningless model outputs or leak DNN model information [2], [3]. The devastating effect on the DNN outputs could lead to serious consequences in safety-critical systems such as smart medical devices or self-driving vehicles.

In general, faults can be injected into a DNN accelerator through different means such as clock glitching [4], [5], voltage glitching [6], dynamic voltage and frequency scaling (DVFS) [7], [8], and Rowhammer [9]. Depending on the type of knowledge required, these attacks can be classified into two groups, those targeting the DNN model and those targeting the underlying hardware. The first group causes misclassification by manipulating some parameters through adversarial attacks or Bit-Flip Attacks [10]–[12]. However, they require complete knowledge of the model (i.e., white-box) to identify the most vulnerable weights or inputs with the highest gradients to attack, which requires a complex search process. In comparison, the second group of attacks, such as clock/voltage glitching, target the hardware accelerators and require very limited information on the model parameters (i.e., black-box). However, faults injected with these methods have limited controllability such that they affect the computation across many classes and hence cause unpredictable faulty outputs.

In this work, we exploit the possibility of performing a *controlled fault injection attack* that derails DNN inputs to a targeted range of classes. More specifically, our study shows that faults injected into different DNN layers have quite diverse impacts. In particular, the fully connected (FC) layers greatly impact inference results, whereas the computation in the FC layers is typically performed in order. Based on this observation, we develop a framework that injects faults via clock glitching and adopts a two-step search process, namely, a coarse-grain search that identifies the time window during which the FC layers are computed, and a fine-grained search to pinpoint the exact classes to attack. This framework is deployed on an FPGA, configured to implement a Deep-learning Processing Unit (DPU) to accelerate three popular DNN models, namely, ResNet50, InceptionV1, and MobileNetV2. To the best of our knowledge, this is the first work that demonstrates a controlled fault injection attack on a black-box DNN.

Overall, the main contributions of this work include:

- The development of a two-step search framework to pinpoint the most vulnerable layers in black-box DNN models;
- Reverse engineering a DNN accelerator to characterize its execution and increase the controllability of fault injection attacks;
- The design and implementation of a controlled fault injection attack that derails arbitrary inputs to a targeted

This work is partially supported by National Science Foundation grant #1909854.

979-8-3503-4859-0/DATE24/© 2024 EDAA

range of classes.

The rest of this paper is structured as follows: Section II briefly reviews existing fault injection attacks on DNN models. Section III analyzes the vulnerabilities of DNN models and the DPU accelerator, which provides a theoretical foundation for the fault injection framework presented in Section IV. Section V presents our case study on the DPU. Finally, Section VI concludes this work.

## II. Background

Fault injection attacks are a class of physical attacks that aim to actively modify the intended behavior of a device to bypass its security or compromise its integrity [13], [14]. These attacks typically target computing architectures that combine CPUs, GPUs, and/or FPGAs by exposing electrical-level security risks [15]. While conventional fault injection attacks target crypto engines, recent work started examining vulnerabilities of DNN hardware accelerators, given their wide usage.

One set of previous works focused on injecting faults into the memory that stores the weights of the DNN model. Givaki et al. [16] developed an attack that undervoltes FPGA on-chip memories. The attack achieved a 30% fault rate in causing memory bit-flips, yet the degradation in model accuracy was only 4.92% because faults were injected randomly into the block RAM, considering the DNN model as a black-box. In comparison, another attack, called DeepHammer [17], leveraged Rowhammer [9] to precisely flip a small set of targeted bits (3 to 24 bits) in memory to degrade the prediction accuracy to random guesses. However, this attack requires complete knowledge of the DNN model (i.e., a white-box attack) so as to apply a gradient-based search to identify a chain of memory bits to flip.

Prior work has also exposed the vulnerability of DNN accelerators by causing timing violations and, consequently, computation errors. The work in [6] proposes a power striker that aggressively overloads the shared Power Distribution Network (PDN) to incur voltage glitches, which in turn induce computation errors or data loading faults. A maximum accuracy drop of 14% on a quantized DNN model was reported. It is worth mentioning that the scheme is a gray-box attack as it requires knowledge of the DNN architecture. Another technique for inducing timing violations is DVFS attack. The attack in [7] sets the working voltage of Nvidia GPUs to a deficient value with software instructions performed on CPU. While this attack is able to degrade inference accuracy by around 70%, it requires complete knowledge of models (white-box) to search for the most sensitive targets.

The work most related to this work is [4], which implements clock glitching on an FPGA that runs a DNN accelerator and uses DSPs for MAC operations. The attack glitches the DSP clock and defines a parameter called glitch intensity as the percentage of the total inference time glitched. However, it requires an intensity as high as 10% to achieve 98% misclassification rate on DNN models such as MobileNet, DenseNet, ResNet50, among others. In comparison, the attack presented in this work only needs to glitch a few clock cycles to cause a high misclassification rate. More importantly, the proposed attack is capable of triggering misclassifications in a small set of target classes.

TABLE I
Summary of different fault injection attacks

| *Work* | Fault injection approach | Platform | Model knowledge | Targeted attack? |
|---|---|---|---|---|
| [16] | Undervolting | FPGA | black-box | No |
| [17] | Rowhammer | CPU | white-box | Yes |
| [6] | Voltage glitching | FPGA | gray-box | No |
| [7] | DVFS | GPU | white-box | Yes |
| [4] | Clock glitching | FPGA | black-box | No |
| This work | Clock glitching | FPGA | black-box | Yes |

Table I summarizes related work and highlights the difference from this work. Overall, to achieve a targeted attack, previous work all requires complete model knowledge, regardless of the platform or the applied fault injection technique. In contrast, this work presents a targeted fault injection attack without knowledge of the model architecture, weights, or hyperparameters.

## III. Fault Vulnerability Analysis

### A. Vulnerabilities of Different DNN layers

A DNN model typically contains various types of layers, including convolutional layers (CONV), activation (typically ReLU), pooling (POOL), fully connected layers (FC), and softmax. These layers are not equally vulnerable to faults. In particular, our analysis shows that *the FC layers are much more vulnerable to faults than others*.

Faults injected into a CONV layer, even if they effectively corrupt the result of a convolution, still have a large chance to be masked, as ReLU removes negative values from an activation map by setting them to zero, while POOL performs downsampling along the spatial dimensions. Moreover, the CONV-ReLU-POOL sequence is typically repeated several times, thus further increasing the chance of dispersing a faulty activation in later layers. In comparison, the FC layer directly computes the scores of different classes, and is typically followed by softmax that transforms the raw outputs into a vector of probabilities. As softmax does not alter the relative importance of each class, fault injected in the FC layer directly affects the prediction outcome. In particular, if faults can be injected in a way that significantly elevates the score of one specific class, that class will become the predicted class.

### B. Vulnerabilities of DPU

Hardware accelerators speed up DNN model execution by increasing the number of computation units running in parallel. Without loss of generality, this work selects the Deep-learning Processing Unit (DPU) [18] as the target architecture. DPU offers three dimensions of parallelism, namely, pixel parallelism (**PP**), input channel parallelism (**ICP**), and output channel parallelism (**OCP**). As illustrated in Fig. 1, PP defines the number of pixels from the feature map that are

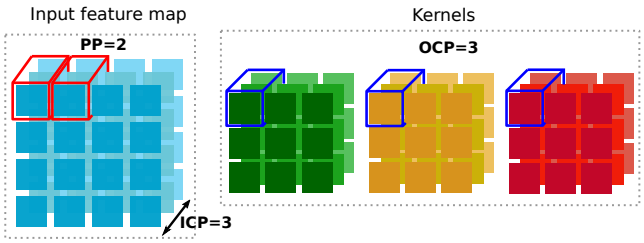



Fig. 1. DPU dimensions of parallelism.

processed simultaneously (red cuboids for PP=2). The ICP and OCP respectively represent the input feature depth and the number of kernels executed in parallel in a convolution operation, and ICP is always equal to the OCP (blue cuboids for ICP=3). The number of operations in a single cycle is defined by 2×PP×ICP×OCP. Since the DPU allows a maximum parallelism with OCP=ICP=16 and PP=8, at most 4096 intermediate results can be computed at a time. Therefore, glitching a few clock cycles can affect a significant amount of intermediate results.

According to DPU specifications, if the FC layer is connected with a flatten layer, the compiler will combine Flatten+FC to a CONV2D layer, and the kernel size is equal to the input feature map size of the flatten layer. Therefore, one can conclude that a short clock glitching injected in the FC layer can potentially affect 2×PP×ICP×OCP intermediate results that contribute to the outputs of **OCP** classes, thus enabling *a targeted attack on the FC layer*.

## IV. Attack Methodology

### A. Threat Model

As mentioned before, the attack presented in this work is a targeted fault injection attack on a black-box DNN model. It is assumed that an adversary can access the hardware accelerator through an interface or share resources, which allows the adversary to inject faults as clock/voltage glitches similar to [4], [6], violating timing constraints. The adversary can control and configure fault injection by setting different parameters via software. Regarding the DNN model, the adversary has no prior knowledge of its architecture or parameters. For example, the adversary can monitor the accelerator via power side-channel to detect when the inference starts. The adversary can also inspect the DNN outputs to choose certain parameters to cause misclassification to specific classes.

### B. Fault Injection Method

The proposed fault injection framework works with any method that can cause timing violations in the target accelerator. Without loss of generality, this work adopts clock glitching, similar to [4]. Specifically, our approach includes two different clock sources at the same frequency but differ in phase (0°/360° and 270°/360°). The computation unit works correctly when only one clock source is used. However, if the clock signal is switched from one source to the other, a glitch is generated. To control the glitch, it is necessary to specify *when* and *how long* it should occur. This is illustrated

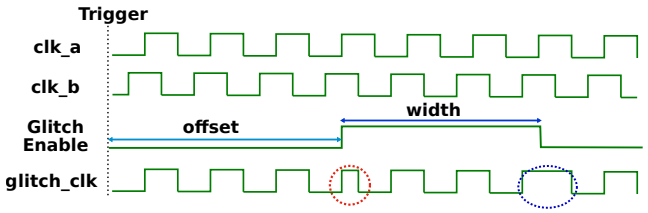



Fig. 2. Clock glitching produced by switching from **clk_a** to **clk_b**

in Fig. 2. Specifically, the proposed setup uses a **trigger** signal to reference the injection. At a specific time point (**offset**), the clock signal is switched from *clk_a* to *clk_b*, generating the glitch (highlighted in red circle) that lasts for a short period (**width**). Therefore, computation errors are expected to occur at the beginning (red circle) and end (blue circle).

### C. Attack Framework

As mentioned before, our attack requires no prior knowledge of the DNN model. To pinpoint the exact classes to attack, the attack framework adopts a two-step search process, namely, a coarse-grain search that identifies the time period corresponding to the FC layer, and a fine-grained search that sets the parameters to attack the targeted classes.

*1) Coarse-grained search:* In this process, faults are injected along the entire inference. The goal is to observe the variation in the overall accuracy so as to identify the start and end of executing the FC layer. A small set of images is randomly selected from the original test dataset to speed up the search. In each run, the **width** parameter is set to a few clock cycles, while the **offset** parameter varies from the beginning to the end of the inference, given a step that determines the search granularity. The outcome of this process is two estimated values: $T_{FCstart}$ and $T_{FCstart}$, which are fed to the second search process.

*2) Fine-grained study:* Compared to coarse-grained search, this stage requires a much larger number of images per class to identify the correlation between the glitch **offset** and the exact classes affected by the injected faults. The **offset** parameter varies between $T_{FCstart}$ and $T_{FCstart}$ (calculated in the previous stage), whereas the **width** parameter is set to a few clock cycles. For each **offset**, the classes with the highest prediction frequency are recorded.

As discussed in Section III-B, the FC layer is calculated sequentially. Therefore, the following equation can be used to calculate the glitch **offset** given a target class to attack:

$$\text{offset} = T_{FCstart} + \frac{C_{target}}{C_{total}} \times (T_{FCend} - T_{FCstart}) \quad (1)$$

where $C_{target}$ is the index of the target class, $C_{total}$ is the total number of classes, and $T_{FCstart}$ and $T_{FCend}$ are the start time and end time of computing the FC layer in the DPU, respectively. Moreover, since multiple classes are processed at the same time, the range of affected classes $R_{target}$ is determined by:

$$R_{target} = \left( \lfloor \frac{C_{target}}{OCP} \rfloor, \lfloor \frac{C_{target}}{OCP} \rfloor + OCP \right) \quad (2)$$

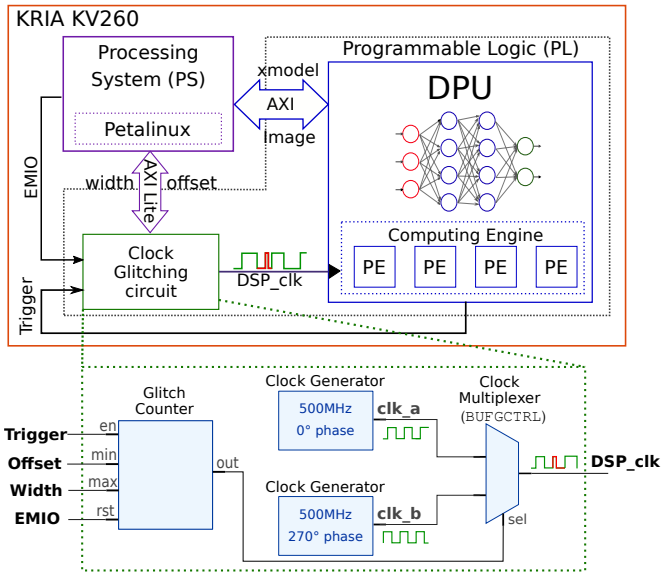



Fig. 3. Experimental setup that implements the DPU and clock glitching circuit on an AMD FPGA board.

where OCP is the number of kernels processed in parallel.

## V. Case Study

This section presents a case study implementing the proposed attack on a Deep-learning Processing Unit (DPU).

### A. Experimental Setup

The DPU is implemented on an AMD KRIA KV260 board, which features a Zynq Ultrascale+ MPSoC device compromising a quad-core ARM Cortex-A53 processor, 256K logic cells, and 1200 DSPs. The DPU has a control unit and a computing engine unit. The latter consists of an array of Processing Engines (PE) that perform MAC operations. Each PE contains DSPs that runs at twice the frequency of the control unit. This frequency is represented as *DSP_clk*, which equals 500MHz in our implementation.

The board combines a Processing System (PS) for software implementations and the Programmable Logic (PL) for hardware implementations. As shown in Fig. 3, the PL is used to implement the DPU and the clock glitching circuit, while the PS communicates with PL through Advanced eXtensible Interface (AXI) protocol, sending the model and input images and receiving the inference outputs. Another AXI interconnect is instantiated to set the **width** and **offset** for the clock glitching circuit, which also receives an EMIO (Extended multiplexed I/O) signal from the PS and a trigger signal from the DPU. The clock glitching circuit switches between two clock generators configured with different phases. The clock multiplexer is instantiated using the primitive `BUFGCTRL`. The glitch counter takes four inputs: EMIO, offset, width, and trigger. Once an attacker enables the glitching circuit by setting the EMIO signal, the trigger signal is automatically activated at the moment when the DPU initiates an inference process. The circuit waits for a certain amount of time (**offset**) and switches the clock source for a short period (**width**).

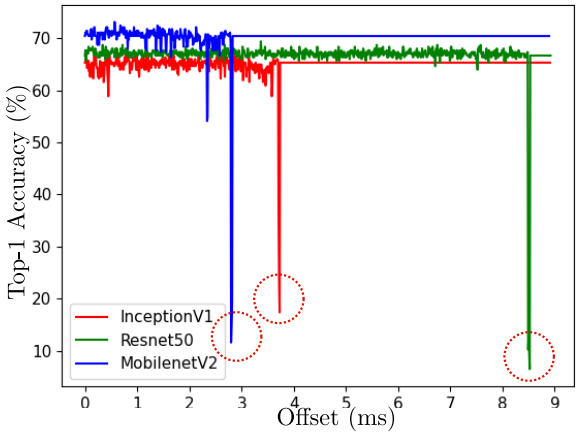



Fig. 4. Overall accuracy drop at different offsets, with maximum drop highlighted.

### B. Models and Dataset

To evaluate the effectiveness of the proposed attack, we have conducted fault injection experiments on three popular DNN models: InceptionV1 [19], ResNet50 [20], and MobileNetV2 [21], all trained on the ImageNet [22] dataset and quantized to 8 bits by Xilinx Vitis AI framework. This tool configures DNN models to run on AMD devices. The experiments use different portions of the test dataset to evaluate the accuracy under the influence of fault injection. Details of the images used in different experiments are provided later.

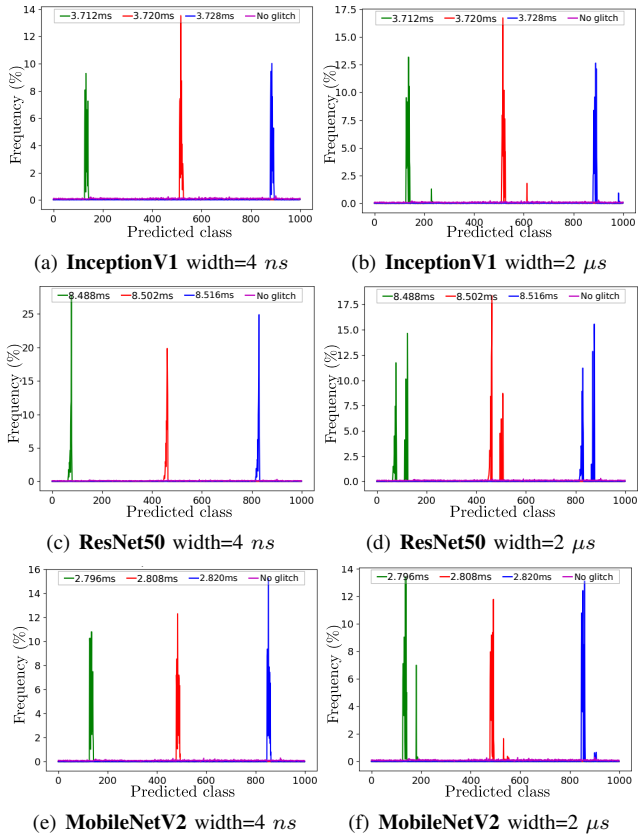
### C. Coarse-grained Search

Our first set of experiments aims to pinpoint the location of the highest accuracy drop (i.e., where FC layer is) in execution. In this study, thousands of experimental runs were conducted, each injecting a single glitch of $4ns$ (i.e., 2 clock cycles) at a different **offset** of the inference process. The **offset** parameter was set to cover the entire inference process. To speed up the search process, 300 images were randomly selected, of which the Top-1 accuracy was calculated for each offset.

Fig. 4 presents the obtained accuracy variation across the entire inference period. The straight line at the end of each model shows that there is no impact on accuracy if the attack is activated after the inference finishes. As can be seen, fault injection causes observable accuracy degradation on all three models. Specifically, the injected faults drop the inference accuracy to 15% on MobileNetV2, 10% on ResNet50, and 20% on InceptionV1 (highlighted by dashed red circles). In all three models, the highest accuracy drop appears towards the end of the inference, confirming that the **FC layer is most vulnerable in all three models**.

### D. Fine-grained Search

Our fine-grained study is carried out to examine the impact of fault on different classes. As an example, Fig. 4 shows that the highest accuracy drop for InceptionV1 occurs when the glitch offset is set to $3.72ms$. Accordingly, in the fine-grained study, the glitch offset is set to a small range covering the highest accuracy drop (e.g., 3.71–3.73$ms$ for InceptionV1),




(a) **InceptionV1** width=4 $ns$ (b) **InceptionV1** width=2 $\mu s$

(c) **ResNet50** width=4 $ns$ (d) **ResNet50** width=2 $\mu s$

(e) **MobileNetV2** width=4 $ns$ (f) **MobileNetV2** width=2 $\mu s$

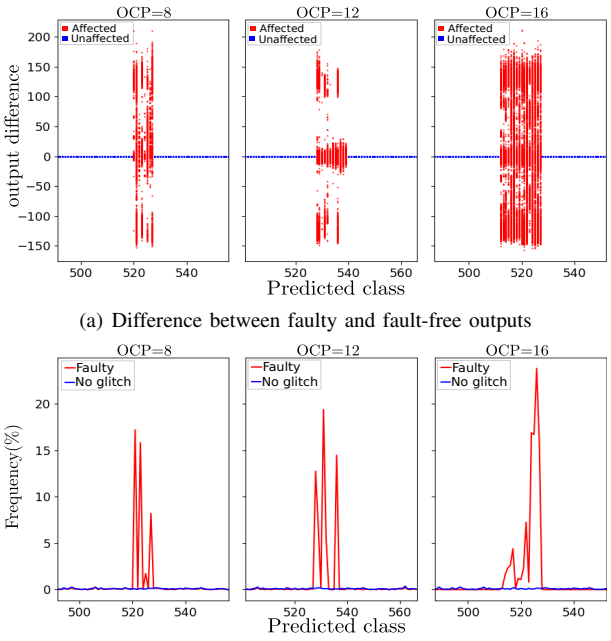
Fig. 5. Distribution of predicted classes. The no-glitch case is flat, while glitches at different offsets shift the output to targeted classes.

TABLE II
PARAMETERS (IN $ms$) FOR TARGETED ATTACK

| *Model* | Inference End | $T_{FCstart}$ | $T_{FCend}$ |
|---|---|---|---|
| InceptionV1 | 3.73045 | 3.70904 | 3.72844 |
| ResNet50 | 8.52280 | 8.48526 | 8.52270 |
| MobileNetV2 | 2.82536 | 2.79188 | 2.82532 |

and the search progresses in much finer steps. The number of test images is increased to 30K (about 30 images per class), while the width ranges from 4$ns$ to 2$\mu s$, to examine its impact.

*1) Targeted attack:* Fig. 5 plots the distribution of predicted classes for all the 30K test images, with and without clock glitching. As shown, the original (no-glitch) distribution is flat, while glitches at different offsets shift the output to targeted classes. For example, on the InceptionV1 model (Fig. 5(a)), glitches were generated at three different offsets: 3.712, 3.720, and 3.728$ms$, which all shift the prediction to a specific range of target classes. A similar trend can be observed for ResNet50 and MobileNetV2 as well. These results confirm, for all three models, the existence of **a linear relationship between the glitch offset and the elevated prediction distribution in certain classes**. As discussed before, this linear relationship is expected since the DPU computes the FC layer in order. A targeted fault injection attack can derive the glitch **offset** with Eq. (1) for a targeted class. For each model, the ($T_{FCstart}, T_{FCend}$) parameters are calculated and listed in Table II.




(a) Difference between faulty and fault-free outputs




(b) Distribution of predicted classes

Fig. 6. Impact of OCP on FC outputs, tested for ResNet-50 with glitch offset=8.502$ms$ and width=20$ns$

*2) Impact of Glitch Width:* As illustrated in Fig. 2, clock glitching is expected to cause computation errors at the beginning and the end of the glitch period. When the glitch width is short (e.g., 4$ns$), the DPU is computing the same set of classes during this period. As a result, only one peak is observed in Fig. 5(a), (c), and (e). In contrast, when the glitch width is increased to 2$\mu s$, the DPU is computing different sets of classes at the beginning and end of the glitch. Therefore, two peaks can be observed in Fig. 5(b), (d), and (f).

*3) Impact of OCP:* As discussed in Section III-B, the DPU computes multiple (=OCP) classes in the FC layer simultaneously. To evaluate its impact, the DPU was configured with three different OCP values: 8, 12, and 16. This study was conducted on ResNet50, while the glitch offset and width are 8.502$ms$ and 20$ns$, respectively.

Fig. 6(a) plots the difference between w/ and w/o glitch in the outputs of the FC layer (before applying softmax). Since the model is quantized to 8-bit, the outputs are in the range of [-128,127]. As expected, OPC defines the number of classes affected by the attack. More importantly, the results show that the difference can be both positive and negative, indicating that clock glitching affects the computation output in both directions. High positive differences typically lead to misclassification into the target classes, whereas negative differences do not alter prediction outcomes in most cases. Overall, OCP=16 is most prone to misclassification, as confirmed in Fig. 6(b).

*4) Success rate:* To demonstrate the effectiveness of the targeted attack, our experiments evaluate its success rate, defined as follows:

$$\text{Success rate} = \frac{\text{\# predictions in } R_{target}}{\text{\# total images}} \times 100 \qquad (3)$$

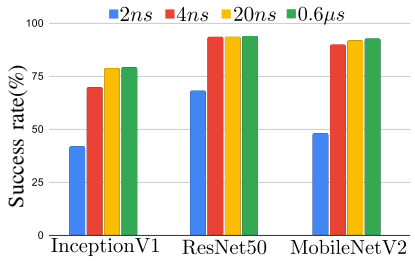



Fig. 7. Success rate of targeted attack.

The ideal success rate of a targeted attack is 100%. That is, any arbitrary image should be predicted as a class in $R_{target}$.

Fig. 7 plots the success rate under different glitch widths: $2ns$, $4ns$, $20ns$, and $0.6\mu s$. These data are the average of the three distinct offset values reported in Fig. 5. As can be seen, a **width** as short as $20ns$ is sufficient to cause a high success rate ($>$75%) on the three models. When the glitch width is small, increasing it leads to a higher success rate. Once it reaches $20ns$, increasing it does not affect the success rate much since computation errors are expected to appear only at the glitch's beginning and end. The results also show that ResNet50 and MobileNetV2 are very vulnerable to the attack, which reaches success rates of **93%** and **92%**, respectively. InceptionV1 is slightly more fault resilient, yet our attack still achieves a **81%** success rate. Overall, these data confirm the effectiveness of the proposed attack.

### *E. Potential countermeasures*

The proposed targeted attack relies on the fact that the FC layer is executed sequentially. One potential mitigation mechanism is partitioning the computation into blocks/tiles, enlarging the range of classes that may have computation errors. This broadens the attack target, however, by making more classes vulnerable. Another possible countermeasure is to add an extra step to check the outputs of the FC layer for error detection and correction. However, standard double and triple modular redundancy methods incur high timing or resource overhead. A method that selectively recomputes FC outputs is desirable.

The proposed targeted attack leverages clock/voltage glitches to cause computation errors. One potential mitigation is the isolation of different voltage/clock domains. A circuit that detects timing violations dynamically could also be added along with the DPU.

## VI. Conclusions

This work demonstrated that DNN hardware accelerators are vulnerable to a targeted fault injection attack, which requires no prior knowledge of the DNN model. The attack is based on the fact that the faults in the FC layer significantly affect DNN outputs, while the computation in the FC layer is performed sequentially. Through a two-step search process, a linear relationship between the clock glitch offset and the targeted set of classes can be established. This attack was implemented in an FPGA and examined for three different DNN models. Experimental results demonstrated a high success rate: 81%, 93%, and 92% for InceptionV1, ResNet50, and MobileNetV2, respectively, achieved with a short glitch of 10 clock cycles. Given its severeness, future work should focus on developing efficient software/hardware countermeasures.

## References

[1] S. Mittal, H. Gupta, and S. Srivastava, "A survey on hardware security of DNN models and accelerators," *Journal of Systems Architecture*, 2021.

[2] F. S. Hosseini, F. Meng, C. Yang, W. Wen, and R. Cammarota, "Tolerating defects in low-power neural network accelerators via retraining-free weight approximation," *ACM Trans. on Embedded Computing Systems*, 2021.

[3] F. Meng, F. S. Hosseini, and C. Yang, "Exploring image selection for self-testing in neural network accelerators," in *ASP-DAC*, 2021.

[4] W. Liu, C.-H. Chang, F. Zhang, and X. Lou, "Imperceptible Misclassification Attack on Deep Learning Accelerator by Glitch Injection," in *DAC*, 2020.

[5] Y. Fukuda, K. Yoshida, and T. Fujino, "Fault Injection Attacks Utilizing Waveform Pattern Matching against Neural Networks Processing on Microcontroller," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, 2022.

[6] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "DeepStrike: Remotely-Guided Fault Injection Attacks on DNN Accelerator in Cloud-FPGA," in *DAC*, 2021.

[7] R. Sun, P. Qiu, Y. Lyu, D. Wang, J. Dong, and G. Qu, "Lightning: Striking the Secure Isolation on GPU Clouds with Transient Hardware Faults," in *arXiv*, 2021.

[8] J. Xu, B. Xuan, A. Liu, M. Sun, F. Zhang, Z. Wang, and K. Ren, "Terminator on SkyNet: a practical DVFS attack on DNN hardware IP for UAV object detection," in *DAC*, 2022.

[9] X. Lou, F. Zhang, Z. L. Chua, Z. Liang, Y. Cheng, and Y. Zhou, "Understanding Rowhammer Attacks through the Lens of a Unified Reference Framework," in *arXiv*, 2019.

[10] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-BFA: Targeted Bit- Flip Adversarial Weight Attack," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2022.

[11] 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*, 2023.

[12] F. S. Hosseini, Q. Liu, F. Meng, C. Yang, and W. Wen, "Safeguarding the intelligence of neural networks with built-in light-weight integrity marks (LIMA)," in *IEEE HOST*, 2021.

[13] S. Koffas and P. K. Vadnala, "On the Effect of Clock Frequency on Voltage and Electromagnetic Fault Injection," in *Applied Cryptography and Network Security Workshops*, 2022.

[14] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, 2006.

[15] D. G. Mahmoud, V. Lenders, and M. Stojilović, "Electrical-Level Attacks on CPUs, FPGAs, and GPUs: Survey and Implications in the Heterogeneous Era," *ACM Computing Surveys*, 2023.

[16] K. Givaki, B. Salami, R. Hojabr, S. M. Reza Tayaranian, A. Khonsari, D. Rahmati, S. Gorgin, A. Cristal, and O. S. Unsal, "On the Resilience of Deep Learning for Reduced-voltage FPGAs," in *28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2020.

[17] F. Yao, A. S. Rakin, and D. Fan, "DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *29th USENIX Security*, 2020.

[18] AMD, "DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide," 2022.

[19] C. Szegedy, L. Wei, J. Yangqing, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE CVPR*, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE CVPR*, 2016.

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *IEEE CVPR*, 2018.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE CVPR*, 2009.