Deep-Dup: An Adversarial Weight Duplication Attack Framework to Crush Deep Neural Network in Multi-Tenant FPGA

Adnan Siraj Rakin, *Arizona State University;* Yukui Luo and Xiaolin Xu, *Northeastern University;* Deliang Fan, *Arizona State University*

https://www.usenix.org/conference/usenixsecurity21/presentation/rakin

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

Deep-Dup: An Adversarial Weight Duplication Attack Framework to Crush Deep Neural Network in Multi-Tenant FPGA

Adnan Siraj Rakin *
Arizona State University
asrakin@asu.edu

Yukui Luo *
Northeastern University
luo.yuk@northeastern.edu

Xiaolin Xu
Northeastern University
x.xu@northeastern.edu

Deliang Fan
Arizona State University
dfan@asu.edu

*Both Authors Contributed Equally

Abstract

The wide deployment of Deep Neural Networks (DNN) in high-performance cloud computing platforms brought to light multi-tenant cloud field-programmable gate arrays (FPGA) as a popular choice of accelerator to boost performance due to its hardware reprogramming flexibility. Such a multi-tenant FPGA setup for DNN acceleration potentially exposes DNN interference tasks under severe threat from malicious users. This work, to the best of our knowledge, is the first to explore DNN model vulnerabilities in multi-tenant FPGAs. We propose a novel adversarial attack framework: *Deep-Dup*, in which the adversarial tenant can inject adversarial faults to the DNN model in the victim tenant of FPGA. Specifically, she can aggressively overload the shared power distribution system of FPGA with malicious power-plundering circuits, achieving *adversarial weight duplication (AWD) hardware attack* that duplicates certain DNN weight packages during data transmission between off-chip memory and on-chip buffer, to hijack the DNN function of the victim tenant. Further, to identify the most vulnerable DNN weight packages for a given malicious objective, we propose a generic vulnerable weight package searching algorithm, called *Progressive Differential Evolution Search (P-DES)*, which is, for the first time, adaptive to both deep learning white-box and black-box attack models. The proposed Deep-Dup is experimentally validated in a developed multi-tenant FPGA prototype, for two popular deep learning applications, i.e., Object Detection and Image Classification. Successful attacks are demonstrated in six popular DNN architectures (e.g., YOLOv2, ResNet-50, MobileNet, etc.) on three datasets (COCO, CIFAR-10, and ImageNet).

1 Introduction

Machine Learning (ML), especially deep neural networks (DNN), services in high-performance cloud computing are gaining extreme popularity due to their remarkable performance in intelligent image/video recognition [1–4], natural language processing [5–7], medical diagnostics [8], malware detection [9], and autonomous driving [10, 11]. Similar to many other high-performance computing (HPC) platforms (e.g., CPU, GPU, ASIC), reconfigurable computing devices like field-programmable gate arrays (FPGA) have been widely deployed in HPC system for DNN acceleration due to their low-effort hardware-level re-programmability to adapt various DNN structures, as well as fast algorithm evolution. For example, IBM and Intel integrated FPGAs in their CPU products for acceleration purposes [12, 13]. Alongside the rapid growth of the cloud computing market and critical developments in DNN hardware acceleration, FPGA has become a significant hardware resource for public lease. Recently, the leading cloud service providers have also started integrating FPGAs into their cloud servers. For example, the Stratix-V FPGA from Intel/Altera has been deployed by the Microsoft Project Catapult for DNN acceleration [14]. Amazon also released its EC2 F1 instances equipped with programmable hardware (UltraScale+VU9P FPGAs) from Xilinx [15].

For high efficiency and performance, there have been growing efforts to support multiple independent tenants co-residing/sharing an FPGA chip over time or simultaneously [16, 17]. The *co-tenancy* of multiple users on the same FPGA chip has created a unique attack surface, where many new vulnerabilities will appear and cause dangerous effects. With many hardware resources being jointly used in the multi-tenant FPGA environment, a malicious tenant can leverage such *indirect* interaction with other tenants to implement various new attacks. However, as a relatively new computing infrastructure, as well as one of the main hardware accelerator platforms, the *security of multi-tenant FPGAs for DNN acceleration* has not been investigated in-depth.

From DNN algorithm point of view, its security has been under severe scrutiny through generating malicious input noise popularly known as *Adversarial Examples* [18–20]. Even though tremendous progress has been made in protecting DNN against adversarial examples [21–23], *neglecting fault injection-based model parameter perturbation does not guarantee the overall security of DNN acceleration in FPGA (DNN-FPGA) system*. Several prior works have effectively demonstrated depletion of DNN intelligence by tempering

model parameters (i.e, weights,biases) using supply chain access [24, 25] or through popular memory fault injection techniques [26–29], which could be in general classified as *adversarial weight attack*. Adversarial weight attack can drastically disrupt the inference behavior towards the intent of a malicious party [26–30]. The large DNN model's parameters (e.g., weights) are extensively tuned in the training process to play a key role in inference accuracy. However, almost all the existing adversarial weight attacks assume an extremely relaxed threat model (i.e., white-box), where the adversary can access all DNN model parameters, like architecture and gradients. Even though it is pivotal to study white-box attacks to understand the behavior of DNN models in the presence of input or weight noise, it is also important to explore how to conduct adversarial weight attacks in a much more strict black-box setup, where the attacker does not know DNN model information.

In summary, **three primary challenges** are **i)** Considering multiple tenants *co-reside* on an FPGA, can a malicious user leverage a novel attack surface to provide the luxury of perturbing DNN model parameters of the victim tenant? **ii)** Can the adversary conduct a black-box adversarial weight attack with no knowledge of DNN model parameters, gradient, etc., instead of white-box attack used in prior works [26, 28]? **iii)** Given an FPGA hardware fault injection attack scheme and a strict black-box threat model, can an adversary design an efficient searching algorithm to identify critical parameters for achieving a specific malicious objective? Inspired by those challenges, we propose *Deep-Dup* attack framework in multi-tenant DNN-FPGA, which consists of **two main modules**: **I)** a novel FPGA hardware fault injection scheme, called *adversarial weight duplication (AWD)*, leveraging two different power-plundering circuits to intentionally inject faults into DNN weight packages during data transmission between off-chip memory and on-chip buffer; **II)** a generic searching algorithm, called *Progressive Differential Evolution Search (P-DES)*, to identify the most vulnerable DNN weight package index and guide AWD to attack for given malicious objective. As far as we know, Deep-Dup is the first work demonstrating that the adversarial FPGA tenant could conduct both untargeted accuracy degradation attack and targeted attack to hijack DNN function in the victim tenant, under both deep learning white-box and black-box setup. The key contributions of this work are summarized as follows:

1): The proposed Adversarial weight duplication (AWD) attack is an FPGA hardware-based fault injection method, leveraging the co-tenancy of different FPGA users, to aggressively overload the shared power distribution system (PDS) and duplicate certain DNN model weight parameters during data transmission between off-chip memory and on-chip buffer. Two different power plundering circuits, i.e., Ring Oscillator (RO) and RO with latch (LRO) are explored and validated in the FPGA attack prototype system.

2): To maximize attack efficiency, i.e. conducting AWD-based fault injection into the most vulnerable DNN weight data packages for any given malicious objective, we propose a generic vulnerable weight package searching algorithm, called *Progressive Differential Evolution Search (P-DES)*. It is, for the *first time*, adaptive to both deep learning white-box and black-box setup. Unlike prior works only demonstrated in a deep learning white-box setup [28], our success in both white-box and black-box mainly comes from the fact that our proposed P-DES does not require any gradient information of DNN model.

3): We are the first to develop an end-to-end Deep-Dup attack framework, one type of adversarial DNN model fault injection attack, utilizing our DNN vulnerable parameter searching software (i.e. P-DES) to guide and search when/where to inject fault through multi-tenant FPGA hardware fault injection (i.e. AWD) for efficient and effective un-targeted/targeted attacks (i.e., un-targeted attack to degrade overall accuracy and targeted attack to degrade only targeted group accuracy).

4): A multi-tenant FPGA prototype is developed to validate the proposed Deep-Dup for two different deep learning applications (i.e., Object Detection and Image Classification). Successful un-targeted and targeted attacks are validated and demonstrated in six different popular DNN architectures (e.g. YOLOv2, ResNet-50, MobileNetV2, etc.) on three data sets (e.g., COCO, CIFAR-10, and ImageNet), under both white-box and black-box setups(i.e. attacker has no knowledge of model parameters (e.g. weights/gradients/ architecture)).

5): As proof-of-concept, our Deep-Dup black-box attack successfully targets the '*Ostrich*' class images (i.e., 100 % attack success rate) on ImageNet with only *20* (out of *23 Million*) weight package fault injection through AWD attacks on ResNet-50 running in FPGA. Besides, Deep-Dup requires just one AWD attack to completely deplete the intelligence of compact MobileNetV2.

2 Background

2.1 Related Attacks on Multi-tenant FPGA

The re-programmability of FPGA makes it a popular hardware accelerator for customized computing [31]. To further explore the advantages of FPGA, leading hardware vendors like Intel and Xilinx have integrated FPGAs with CPUs [13] or ARM cores to build flexible System-on-Chips (SoCs) [32, 33]. These heterogeneous computing platforms have recently been integrated into cloud data centers [34], where the hardware resources are leased to different users. The *co-tenancy* of multiple users on the same FPGA chip, although improves the resource utilization efficiency and performance, but also creates a unique attack surface, where many new vulnerabilities will appear and cause dangerous results. With many critical hardware components (e.g., power supply system) being jointly used in the multi-tenant FPGA environment, a malicious tenant can leverage such *indirect* interaction with other tenants to implement various new attacks.

Generally, the attacks on multi-tenant FPGAs can be classified into two classes: 1) side-channel attack, in which the adversarial FPGA user can construct hardware primitive as sensors(e.g., ring oscillator (RO)), to track and analyze the secret of victim users. For example, in [34], the RO-based sensor used as power side-channel has successfully extracted the key of RSA crypto module, similarly, key extraction from advanced encryption standard (AES) is successfully demonstrated in [35] based on RO-caused voltage drop. More recently, it has been demonstrated that a malicious user can leverage the crosstalk between FPGA long-wires as a remote side-channel to steal secret information [36, 37]. 2) Fault injection attack, in which the adversary targets to inject faults to or crash the applications of victim users. For example, the entropy of true random number generator is corrupted by power attacks in multi-tenant FPGAs [38]. In [39], the aggressive power consumption by malicious users causes a voltage drop on the FPGA, which can be leveraged to introduce faults.

With Machine Learning as a service (MLaaS) [40, 41] becoming popular, public lease FPGAs also become an emerging platform for acceleration purposes. However, the security of using multi-tenant FPGA for DNN acceleration is still under-explored in existing works, which is the main target of this paper. Specially, the proposed Deep-Dup methodology belongs to the fault injection category, which leverages malicious power-plundering circuits to compromise the integrity of the DNN model for un-targeted or targeted attacks.

2.2 Deep Learning Security

There has been a considerable amount of effort in developing robust and secure DL algorithms [18, 19, 22, 25, 42–49]. Existing deep learning attack vectors under investigation mainly fall into three categories: *1)* Attacks that either mislead prediction outcome using maliciously crafted queries (i.e., adversarial inputs/examples [22, 50]) or through miss-training the model with poisoned training set (i.e., data poisoning attacks [51, 52]). *2)* DL information leakage threats such as membership inference attacks [49, 53] and model extraction attacks [47, 54] where adversaries manage to either recover data samples used in training or infer critical DL model parameters. *3)* Finally, adversarial fault injection techniques have been leveraged to intentionally trigger weight noise to cause classification errors in a wide range of DL evaluation platform [26–29, 55].

The first two attacks are generally considered as *external adversaries* that exploit training and inference inputs to the deep learning model. Despite the progress in protecting DNN against this external adversaries [21–23], neglecting internal adversarial fault injection still puts the overall security of DNN acceleration in FPGA (DNN-FPGA) systems under threat. The most recent adversarial weight attacks [27, 28, 30, 56] demonstrated, in both deep learning algorithm and real-word general-purpose computer system, that it is possible to modify an extremely small amount (i.e., tens out of millions)

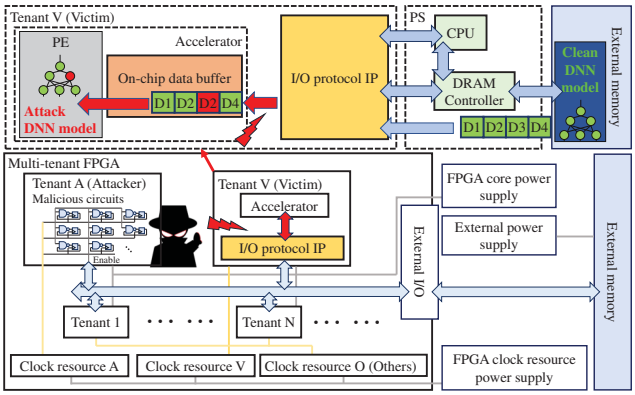


Figure 1: Threat model for the proposed Deep-Dup.

of DNN model parameters using row-hammer based bit-flip attack in computer main memory to severely damage or hijack DNN inference function. Even those injected faults might be minor if leveraged by a malicious adversary, such internal *adversarial fault injection* harnessing hardware vulnerabilities may be extremely dangerous as they can severely jeopardize the confidentiality and integrity of the DNN system.

3 Threat Model and Attack Vector

Multi-tenant FPGA Hardware Threat Model. In this work, we consider the representative hardware abstraction of multi-tenant FPGA used in the *security* works [36, 57, 58], and *operating system* works [17, 59]. The threat model is shown in Fig. 1, which has the following characteristics: (1) Multiple tenants *co-reside* on a cloud-FPGA and their circuits can be executed simultaneously. The system administrator of cloud service is trusted. (2) Each tenant has the flexibility to program his design in the desired FPGA regions (if not taken by others). (3) All tenants share certain hardware resources on an FPGA chip, such as the PDS and the communication channels with external memory or I/O. (4) We assume that the adversary knows the type of transmitted data (i.e., either DNN model or input data) on the communication channel (e.g., I/O protocol IP) connecting the off-chip memory and on-chip data buffer. Adversarial FPGA tenants can learn such information in different ways: i) Using the side-channel leakage from the communication/data channels on the FPGA, e.g., the cross-talk between FPGA long-wires [36]. Besides, recent works have reverse engineered DNN using side-channel attacks to practically recover its information (i.e, architecture, weights) [60, 61]. Additionally, it is practical to recover the DNN model using instruction flow leakage [62]. ii) Practically, the victim FPGA tenant can be the provider of Machine learning as a service (MLaaS) [40, 41], who offer accelerated DNN computation on multi-tenant FPGA, and the adversary can rent such service as a normal customer, then he/she can learn some info of the model and query outputs. More importantly, our black-box attack only requires to know the transmitted data type (i.e. weight or input), instead of

actual weight values, which is recoverable using similar methods as in [36, 60, 61]. It is worth mentioning that, although the current cloud-computing business model has not yet supported simultaneous resource-sharing, with the significant development of FPGA-based cloud computing, e.g., dynamic workload support [59], FPGA virtulization [63], multi-tenant FPGA is envisioned to be possible in the future [64].

Deep Learning (DL) Algorithm Threat Model. Regarding the Deep Learning algorithm level threat model, in this work, following many prior DL security works [18, 21, 26–28, 56, 65, 66], two different DL algorithm threat models are considered and defined here: 1) *DL white-box*: attacker needs to know model architectures, weight values, gradients, several batches of test data, queried outputs. 2) *DL black-box*: attacker only knows the queried outputs and a sample test dataset. Unlike the traditional DL white-box threat model [18, 21, 27, 67], our DL white-box is even weaker with no requirement of computing gradient during the attacking process. Since different DL security works may have different definitions of white/black-box, throughout this work, we will stick to the definition here, which is commonly used in prior works [27, 67, 68]. In this work, similar to many adversarial input or weight attacks, we only target to attack a pre-trained DNN inference model in FPGA, i.e., hijacking the DNN inference behavior through the proposed Deep-Dup, not the training process, which typically requires extra access to the training supply chain [24, 69].

In our threat model defined in Fig. 1, the adversary will leverage our proposed AWD based fault injection attack on the weight packages identified by our proposed P-DES searching algorithm, when transmitting the DNN model from off-chip memory to on-chip buffer/processing engine (PE), resulting in a weight perturbed DNN model in the PEs. After the attack, the DNN function is hijacked by an adversary with malicious behaviors, such as accuracy degradation or wrong classification of a targeted output class.

4 Attack Objective Formulation

The proposed Deep-Dup attack is designed to perform both un-targeted and targeted attacks, defined as below.

Un-targeted Attack. The objective of this attack is to degrade the overall network inference accuracy (i.e., misclassifying whole test dataset), thus maximizing the inference loss of DNN. As a consequence, the objective can be formulated as an optimization problem:

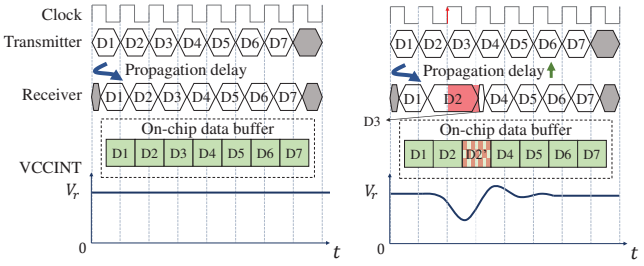
$$\max \ \mathcal{L}_u = \max_{\{\hat{W}\}} \ \mathbb{E}_{\mathbb{X}} \mathcal{L}(f(\boldsymbol{x}, \{W\}); \boldsymbol{t}) \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{t}$ are the vectorized input and target output of a given test batch and $\mathcal{L}(\cdot,\cdot)$ calculates the loss between DNN output and target. The objective is to degrade the network's overall accuracy as low as possible by perturbing weights of the clean DNN model from W to $\hat{W}$.

Targeted Attack. Different from the un-targeted attack, the objective of targeted attack in this work is to misclassify a specific (target) class of inputs (t_s). This attack objective is formulated in Eq. 2, which can be achieved by maximizing the loss of those target class:

$$\max \ \mathcal{L}_t = \max_{\{\hat{W}\}} \ \mathbb{E}_{\mathbb{X}} \mathcal{L}(f(\boldsymbol{x}_s, \{W\}); \boldsymbol{t}) \tag{2}$$

where $\boldsymbol{x}_s$ is a sample input batch belongs to the target class t_s.



(a) DNN model transmission w/o attack.

(b) DNN model transmission under AWD attack.

Figure 2: Illustrated timing diagrams of DNN model transmission w/o or under AWD attack. (a) Each DNN weight package (Di) is transmitted and received in a separate clock cycle. (b) Voltage glitch incurs more propagation delay to the transmission of D2, which also shortens the next package D3. As a result, the data package D2 is sampled twice by the receiver clock, injecting faults to the received data package.

5 Proposed Deep-Dup Framework

Deep-Dup mainly consists of two proposed modules: 1) *adversarial weight duplication (AWD)* attack, a novel FPGA hardware fault injection scheme leveraging power-plundering circuit to intentionally duplicate certain DNN weight packages during data transmission between off-chip memory and on-chip buffer; 2) *progressive differential evolution search (P-DES)*, a generic searching algorithm to identify most vulnerable DNN weight package index and guide AWD fault injection for given malicious objective. In the end of this section, we will present Deep-Dup as an end-to-end software-hardware integrated attack framework.

5.1 AWD attack in multi-tenant FPGA

5.1.1 Preliminaries of DNN model implementations

The schematic of an FPGA-based DNN acceleration is illustrated in Fig. 1, consisting of a processing system (PS), processing engine (PE), and external (off-chip) memory. Practically, DNN computation is usually accomplished in a *layer-by-layer* style, i.e., input data like image and DNN model parameters of different layers are usually loaded and processed separately [70–72]. Fig. 1 shows the flow of FPGA I/O protocol IP for typical DNN model transmission, in which the on-chip data buffer sends a data transaction request to PS for loading data from external memory. Then, the processing engine (PE) will implement computation based on the DNN model in the on-chip data buffer (e.g., BRAM).

A data transmission flow is shown in Fig. 2a, in each clock cycle, a *data package* (D*i*) is transmitted from transmitter (e.g. external memory) to receiver. Taking the advanced eXtensible interface4 (AXI4) as an example [73], the receiver first sends a data request with an external memory address, and then it will be notified to read the data when it is ready. The size of each transmitted data package depends on the channel bandwidth. In DNN model transmission, the normal (w/o attacks) transmission flow with each D*i* as a DNN weight package is illustrated in Fig. 2a, with FPGA core voltage (`VCCINT`) being stable at the recommended supply voltage (V_r), N data packages (e.g., weights) are transmitted in N clock cycles (D1-D7 in Fig. 2a).

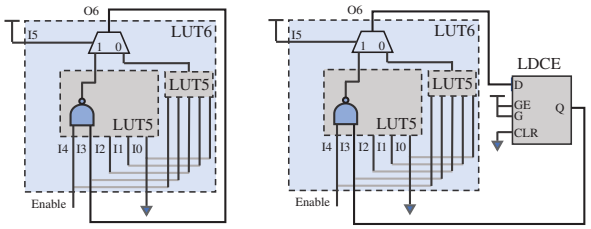
5.1.2 AWD based fault injection into DNN model

The power supply of modern FPGA chips is regulated based on their voltages, different components will be activated following the order of their nominal voltage, e.g., from low to high [74–76]. Most FPGAs utilize a hierarchical power distribution system (PDS) [1], which consists of some power regulators providing different supply voltages [75, 76, 78]. A critical component of PDS is the capacitor used as the "power bank" for the operational reliability of FPGA. For example, when an FPGA chip's power supply is suddenly overloaded (i.e., by a transient higher power demand), these capacitors are discharged to compensate for the extra power that regulators cannot immediately provide. The capacitors of FPGA PDS are usually sized accordingly to fit the practical need. Formally, the default output capacitance (C_{out}) of an FPGA is usually sized to compensate for the current difference for at least two clock cycles with a tolerable voltage drop [78]. As calculated in Eq. 3, where ΔI_{out} and ΔV_{out} represent the changes of output current and voltage, respectively, and f_{sw} denotes the regulator switching frequency.

$$C_{out} = \frac{2 \times \Delta I_{out}}{f_{sw} \times \Delta V_{out}} \tag{3}$$

As one of FPGA's most critical parameters, the clock signals provide standard and global timing references for all on-chip operations. In practice, to generate different timing signals, i.e., with different frequencies or phases, FPGAs are equipped with several clock management components, such as the phase-lock-loop. The on-chip clock signals are usually generated by various clock management components, and their reliability is heavily dependent on the robustness of these components. To enhance clock integrity, these clock components are powered by separate supply voltage resources (Fig. 1) from the computing elements like PE. For example, the clock components of Xilinx FPGAs are powered by the auxiliary voltage `VCCAUX` rather than the FPGA core supply voltage `VCCINT` [79]. Such a separate power supply mechanism ensures sufficient energy for the operation of these clock components, thus enhancing reliability.

[1] PDS is the official terminology of Xilinx FPGAs, while Intel FPGAs use power distribution networks [77]. For uniformity, we use PDS in this paper.



(a) A power-plundering cell based on ring-oscillator (RO).

(b) A cloud-sanctioned power-plundering cell based on RO with a Latch (LRO).

Figure 3: Two power-plundering circuit examples on FPGA

.

The DNN execution in FPGA is significantly relying on the integrity of its loaded model. Our proposed AWD attack is motivated by two facts: 1) As aforementioned, the reliability and correctness of FPGA applications are ensured by the power delivery mechanism; 2) Based on the power regulation mechanism, there exists a *maximum power capacity* that FPGA PDS can provide to PEs. Thus, if the FPGA PDS is overloaded, FPGA applications might encounter faults caused by the timing violation between the clock signal and computation/data. Recent works have demonstrated that the activation of many power-plundering circuits (e.g., ROs), can cause transient voltage drop on the FPGA [35, 38, 80], thus incurring fault injection.

Considering the importance of frequent and real-time DNN model transmission from/to FPGA, the basic idea for AWD attack is that a malicious FPGA tenant can introduce a timing violation to the DNN model transmission from off-chip memory to the on-chip data buffer. As illustrated in Fig. 2a, a stable FPGA core voltage (`VCCINT`) (i.e., with trivial or no fluctuations) will not cause timing violations to data transmission. However, an unstable `VCCINT` will incur serious timing violations. For example, a sudden voltage drop will make the digital circuit execution slower than usual, causing a longer propagation delay to the data transmission. As shown in Fig. 2b, the adversary's aggressive power plundering creates a voltage drop/glitch that incurs slowing down the data transmission channel. As a result, the corresponding data package (e.g., D2) may be sampled twice by the receiver clock, causing a fault injection into the following data package. We envision that maliciously designed fault-injected weight data packages will greatly impact the DNN computation, inducing either significant performance loss, or other malicious behaviors.

5.1.3 Power-plundering circuits

A power-plundering circuit can be achieved with any circuit scheme with high dynamic power consumption, e.g., ring-oscillator (RO) circuits. However, it should be noted that although RO circuit provides high power-plundering potential, it can be possibly detected by the FPGA development tools [81]. To make power-plundering more stealthy, i.e., cloud-sanctioned, some recent works employ common FPGA applications, e.g., the shift registers of an AES circuit [16]

and XOR tree circuit [82]. Since this work focuses on the security of the DNN model in multi-tenant FPGA, we adopt two power-plundering schemes, RO and Latch RO (LRO), for proof-of-concept. Fig. 3a shows the RO circuit instantiated with an FPGA look-up table (LUT). Different from RO, the LRO circuit shown in Fig. 3b has a latch in the loop, which is a cloud-sanctioned design scheme that can bypass the design rule checking for combinational loop in FPGA design tools. In detail, these two power-plundering schemes are both instantiated as a NAND gate controlled by an `Enable` signal. An adversarial FPGA tenant can employ a large number of such cells controlled by the same `Enable` signal, which can be activated to overload the FPGA PDS and introduce transient voltage drop shown in Fig. 2b, thus implementing fault injection attack. Note that the proposed attack in this paper can be achieved with any other cloud-sanctioned power plundering design, such as the AES-based scheme in [16].

5.1.4 AWD attack triggering system

As mentioned in the hardware threat model (Sec.3), our proposed attack only requires the adversary to know the type of data (i.e., weight or not) being transmitted on the FPGA and the **starting/ending points**, which can be achieved with side-channel (e.g., power) analysis. To demonstrate this, we build the AWD triggering system with two major components: ① *Time-to-Digital Converter* (TDC) based sensor and ② *Triggering BRAM*, as shown in Fig. 4. We prototype a TDC circuit in FPGA to capture the on-chip voltage fluctuation and measure the digital output of the TDC sensor during the execution of DNN (YOLOv2 in this example). We observe a strong correlation between the sensor outputs and DNN execution, i.e., weight transmission or functional layers' execution. For example, as shown in Fig. 4, the TDC sensor outputs corresponding to weight transmission periods are relatively stable (i.e., much less voltage fluctuation), since it consumes much less power than the functional layers, like Max pool or Convolution. Due to the page limit, we omit the TDC sensor design details and refer interested readers to the related work [83] for details.

Based on the TDC sensor output, we profile a *triggering strategy file* to control the AWD attack activation, which consists of three parameters: triggering delay, triggering period, and target index. The strategy file is stored in the triggering BRAM (②), composed of '1s' and '0s, which are used to activate or disable the power-plundering circuit, respectively. With the triggering BRAM being read at a certain clock frequency, this system can control the triggering of fault injection. For example, a series of consecutive '0s' disable the power plundering circuit for a certain time period, while a series of consecutive '1s' defines the length of the attack period. By selecting the locations of '1s', we can choose to inject faults on specific DNN weights of specific attack indexes obtained from our P-DES searching algorithm (Sec.5.2).

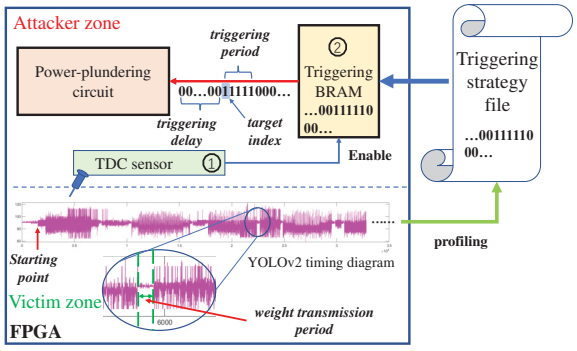


Figure 4: AWD triggering system. A TDC sensor is used to capture voltage fluctuation during the YOLOv2 execution, in which the weight transmission period can be clearly observed.

5.2 P-DES Searching Algorithm

This section delineates the proposed vulnerable weight searching algorithm, called *Progressive Differential Evolution Search (P-DES)*, to generate a set of weight data package index for AWD to attack, given attack objective. To formally define the problem, let us first consider a L layer network with weight parameters-$W_{l=1}^{L}$. Then, the after-attack (i.e. perturbed) weight of the target DNN model executed in FPGA will become $\hat{W}_{l=1}^{L}$. We model different attack objectives aiming to minimize the difference between $W_{l=1}^{L}$ and $\hat{W}_{l=1}^{L}$ for deriving the minimal number of required AWD attacks performing both defined un-targeted and targeted attack objectives.

To clearly describe the searching algorithm, we start from modeling of white-box attack, assuming attacker knows the exact model parameters (i.e. weight values and architecture). The black-box attack will leverage a similar searching algorithm and its corresponding adaption will be described in the end-to-end attack framework section. We assign each weight package in the target DNN with two indexes (p,q); where p denotes the layer index and q denotes the index of weight at layer p after flattening the weight matrix $\mathbf{W}$ ($\mathbf{W} \in R^{m\times n\times a\times kw}$) into a 1D array. Note that, here the weight package refers to one data package that is transmitted in one clock cycle. In the following, we may just call it weight for simplification. The proposed search algorithm is general and applicable for both attack objectives described in Sec. 4.

P-DES is a progressive search algorithm integrating with the concept of differential evolution [84–86]. The goal is to progressively search for one weight index at each iteration to guide AWD attack until the attacker-defined malicious objective is satisfied. The flow chart of the proposed P-DES is shown in Fig. 5. For $n^{th} iteration$, it starts by initializing a set of random weight candidates (i.e. population set - $\mathbf{S}$) for attacker to perform AWD attack and evaluate each attack effect (i.e. fitness function) at current iteration. Then it runs through a succession of evolutionary steps: *mutation*, *crossover* and *selection* for z times (known as the number of *evolution*, '500' in this work) to gradually replace original candidates with bet-

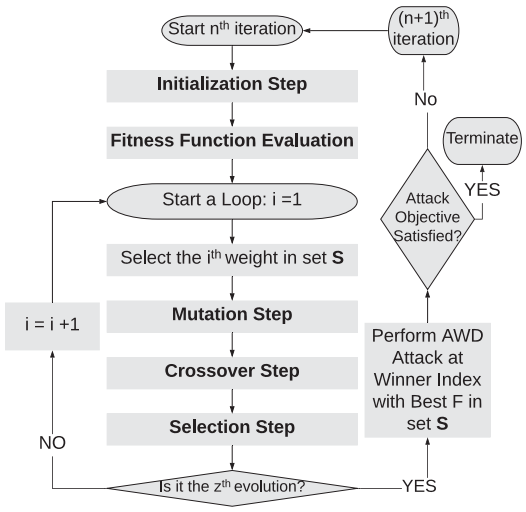


Figure 5: Overview of proposed adversarial weight index searching (P-DES) algorithm.

ter ones for achieving the attacker defined malicious objective. When z times evolution is finished in one search iteration, the attacker picks one best candidate (weight index with highest fitness function value- F) among the final survived population set S and conduct an AWD attack on this winner weight location to duplication data package as described in the previous sub-section. The detailed description of each step is as follow:

Initialization Step. As described above, the objective of differential evolution is to improve population set **S** over time to gradually reach the attacker-defined malicious objective. To initialize, **S** will start with a set of random values, containing z weights whose indexes located at (p_l, q_l) ; where $l = 1,2,3,..,z$. Here, z is the size of **S**, defined as the number of evolution. Ideally, a larger population set (i.e., higher z) would result in a better attack performance at the cost of increased searching time.

Fitness Function Evaluation. Fitness function - F_l is an important step of an evolutionary algorithm to evaluate the attack effect of each proposed candidate in the population set **S**. In our Deep-Dup attack, as defined in Eq. 1 and Eq. 2, we assign the DNN loss function as fitness function. Thus we could evaluate the attack effect (i.e. F_l) of each candidate in set **S** in terms of DNN loss. Note that, for a white-box attack, such evaluation (i.e. fitness function) could be computed in an off-line replicated model. For black-box attack, the loss will be directly evaluated in FPGA by conducting AWD attack in the proposed candidate index pointed data package clock. In the next sub-section, a detailed Deep-Dup framework for both white-box and black-box attacks will be discussed. In P-DES, the attacker's goal is to maximize the fitness function - F_l to achieve un-targeted (Eq. 1) or targeted attack (2):

$$F_l \in \{\mathcal{L}_u, \mathcal{L}_t\} \tag{4}$$

where L_u is un-taregeted attack loss and L_t is targeted attack loss. Note that, the after each evaluation of F_l, attacker needs to restore the original weight values W by reloading the weights, to guarantee each fitness function is evaluated only based on one corresponding attack weigh index.

Mutation Step. For each weight index candidate in population set S, the mutation step generates new candidates using specific mutation strategy to improve current population set. In this work, we integrate four popular mutation strategies [87, 88], where each one generates one mutant vector. Thus, a mutant vector ($\{p_{mut}, q_{mut}\}$ $=\{(p_{mut1}, q_{mut1});(p_{mut2}, q_{mut2});(p_{mut3}, q_{mut3});(p_{mut4}, q_{mut4}\}$)) is generated for each weight index candidate:

Strategy 1:

$$p_{mut1} = p_a + \alpha_1(p_b - p_c); \tag{5}$$
$$q_{mut1} = q_a + \alpha_1(q_b - q_c) \tag{6}$$

Strategy 2:

$$p_{mut2} = p_a + \alpha_1 \times (p_b - p_c) + \alpha_2 \times (p_d - p_e); \tag{7}$$
$$q_{mut2} = q_a + \alpha_1 \times (q_b - q_c) + \alpha_2 \times (q_d - q_e) \tag{8}$$

Strategy 3:

$$p_{mut3} = p_a + \alpha_1(p_{best} - p_a) + \alpha_2(p_b - p_c) + \alpha_3(p_d - p_e); \tag{9}$$
$$q_{mut3} = q_a + \alpha_1(q_{best} - q_a) + \alpha_2(q_b - q_c) + \alpha_3(q_d - q_e) \tag{10}$$

Strategy 4:

$$p_{mut4} = p_a + \alpha_1(p_{best} - p_{worst}); \tag{11}$$
$$q_{mut4} = q_a + \alpha_1(q_{best} - q_{worst}) \tag{12}$$

where $\alpha_1, \alpha_2, \alpha_3$ are the mutation factors sampled randomly in the range of [0,1] [87]. a, b, c, d, e are random numbers ($a \neq b \neq c \neq d \neq e$) generated in the range of [0,z]. (p_{best}, q_{best}) and (p_{worst}, q_{worst}) are the indexes with the best and worst fitness function values. Note that, both p and q for each layer are normalized to the range of [0,1], which is important since the amount of weights at each layer is different.

Crossover Step. In the crossover step, attacker mixes each mutant vector (p_{mut}, q_{mut}) with current vector (p_i, q_i) to generate a trial vector(p_{trail}, q_{trial}):

$$if\ p_{mut} \in [0,1]:\ p_{trial} = p_{mut};\quad else:\ p_{trial} = p_i \tag{13}$$
$$if\ q_{mut} \in [0,1]:\ q_{trial} = q_{mut};\quad else:\ q_{trial} = q_i \tag{14}$$

The above procedure guarantees attacker only chooses the mutant feature with a valid range of [0,1]. Then, the fitness function is evaluated for each trial vector (i.e., F_{trial1},F_{trial2},F_{trial3},F_{trial4}). This crossover step ensures the attacker can generate a diverse set of candidates to cover most of the DNN weight search space.

Selection Step. The selection step selects only the best candidate (i.e. winner with the highest fitness function value) between the trial vector set ($\{p_{trial}, q_{trial}\}$ with four trial vectors) and current candidate (p_i, q_i). Then, the rest four will be eliminated. The above discussed mutation, crossover and selection will repeat z times to cover all candidates in the population set S. As a result, the initial randomly proposed S will evolve over time to gradually approach the attacker-defined malicious objective. When z times evolution is finished, the attacker could perform AWD attack at the winner (with the highest fitness function value in S) weight package during transmission. P-DES will check if the attack objective has been achieved. If yes, it stops. If not, it goes to the next iteration for a new round of attack iteration.

5.3 End-to-End Deep-Dup Attack

This sub-section discusses the proposed end-to-end Deep-Dup attack framework integrating *training* software (i.e. searching) utilizing P-DES algorithm and hardware fault injection through AWD, i.e. fault *triggering*. We also experimentally demonstrate the success of our end-to-end attack framework from the attacker's input end to the victim's output end for white-box and black-box attack. Note that, the fault injection reliability (i.e. fault injection success rate) and detection analysis will be discussed in detail in the experimental section 7.1 and 7.5. The main mechanism of our Deep-Dup attack framework could succeed even with real-world un-reliable hardware fault injection (i.e., with probability to succeed) is based on the fact that the vulnerable weight sets that our P-DES searching algorithm identifies are not static or unique, meaning the targeted attack index set could be progressively expanded based on real measured attack effect, for the same malicious objective. This is possible due to that deep learning model parameter training is a high dimension optimization process and many different fault injection combinations could lead to the same effect, which is also observed in prior works [27, 28, 89]. Thus, our proposed progressive evolutionary searching algorithm could take care of such fault injection uncertainty and randomness through redundant attack iterations to greatly improve the overall attack success rate, which is also experimentally validated in Sec.7.3 and 7.5.

5.3.1 White-Box Attack Framework

Training through P-DES. As we discussed in the threat model, white-box attack assumes adversary knows all the details of target DNN model in victim FPGA, including architecture, weight values, gradients, weight package transmission over FPGA I/O protocol IP. . As shown in Fig. 6, knowing these execution details of the target DNN model, the adversarial can build an *off-line simulator* (i.e. model replicate) to emulate the execution of target DNN in FPGA. Meanwhile, prior profiling should be conducted to estimate the fault injection success rate f_p (84.84% and 58.91% for our measured RO and LRO based power plundering circuits), which will add randomness to the off-line simulated fault-injected DNN model and thus the fitness function evaluation (Eq.4). Note that, this f_p does not need to be very accurate. In general, smaller f_p will force the progressive P-DES algorithm to generate a more redundant attack index to compensate for higher uncertainty of fault injection. More experiment results demonstrating the co-relation between f_p and attack iterations are provided in Sec. 7.5 (Tab. 5). With the help of this off-line simulator, the P-DES searching algorithm will generate the attack index ①, i.e. model weight package index to be attacked during data communication.

Triggering AWD. In the next step ②, the P-DES generated attack index will be sent to our AWD triggering system to implement actual fault injection on those locations to achieve the defined malicious objective. More details of triggering system implementation are described in Sec.5.1.4. To summarize, the attacker profiles the targeted DNN weight package indexes through the TDC sensor and embeds the received attack index from the last step into the attacking strategy file (Fig. 4), which automatically triggers and controls the power-plundering circuits to implement the fault injection in the designed locations. After that, if the attack objective is not achieved (i.e., due to un-successful fault injection), the attacker will repeat the steps ① and ② to re-generate a more redundant attack index until successful.

5.3.2 Black-Box Attack Framework

Fig. 7 shows the overview of Deep-Dup black-box attack framework. Instead of constructing an off-line replicate to search vulnerable weights in white-box attack, in black-box attack, Deep-Dup directly utilizes run-time victim DNN in target FPGA to evaluate the attack effectiveness (i.e. fitness function) of our searching algorithm P-DES proposed weight candidate in mutation step for every attack iteration. Thus, the un-reliable fault injection phenomenon is automatically considered and evaluated in the framework since the fitness function is directly evaluated in the victim FPGA using the real fault injection attack.

In this black-box setting, for every attack iteration, the attacker first utilizes the mutation function defined in our P-DES algorithm to propose a potential attack index candidate ①. Next, it will be sent to the AWD triggering component (Fig.4) to implement fault injection ② in current evolution. Therefore, the current DNN model in FPGA is executed based on the fault-injected model, where its DNN output ③ will be read out by the attacker to be recorded as attack effectiveness (i.e. fitness function evaluation). Note that, during this process, the fault injection may succeed, or not. As for an attacker, since it is a black-box, he/she does not know about it. Only the victim DNN output response w.r.t. currently proposed attack index will be recorded and sent back to our P-DES software. Then, this step ①-②-③ will repeat z evolution times to select one winner attack index to finish the current attack iteration. After that, a new attack iteration will be started to find the next winner attack index until the defined attack objective is achieved.

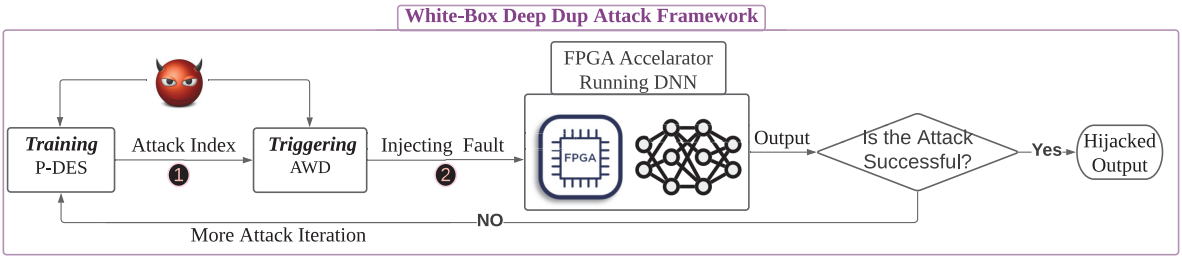


Figure 6: Overview of End-to-End Deep-Dup attack framework integrating P-DES and AWD for White-Box attack

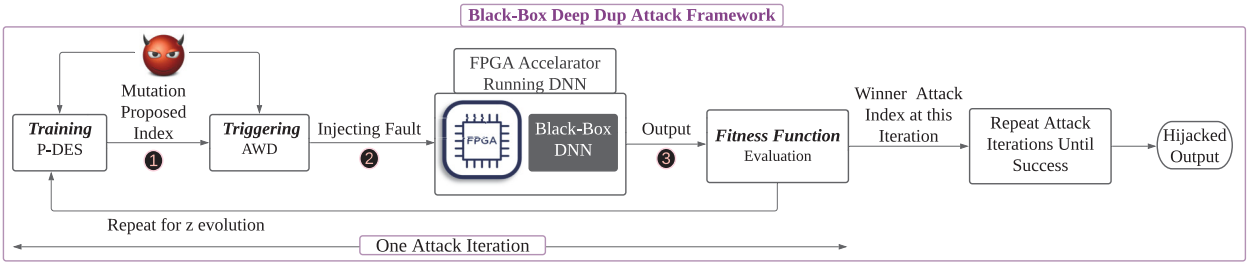


Figure 7: Overview of End-to-End Deep-Dup attack framework integrating P-DES and AWD for Black-Box Attack.

Modification of P-DES to adapt to Black-Box. For a black-box attack, the attacker can only access the input and output scores of the target DNN in victim tenant FPGA, with no knowledge of DNN architecture (i.e., in P-DES, p refers to # of layers & q refers to # of weights at each layer) (details in section 5.2). To adapt the P-DES algorithm to a black-box attack, instead of using architecture info of p and q (i.e., 2D vector), we will treat the whole network parameter to be unwrapped into a 1D vector w, where an attacker tries to identify each weight with one feature $\hat{p}$. Here, $\hat{p}$ denotes the weight index to be attacked after flattening and combining all L layers weights sequentially. As we defined in the threat model section and AWD triggering section (sec.5.1.4), this is feasible since the attacker knows which clock cycles are used to transmit DNN model weights, enabling an attacker to develop such a 1D weight index vector for the P-DES. This is the only modification needed for P-DES algorithm discussed in section 5.2 to adapt to black-box attack.

Triggering AWD in Black-Box. Most of the AWD triggering scheme (details in Sec.5.1.4) of black-box attack is similar to that in white-box (i.e., controlled by the attacking strategy file), except that it will be triggered much more frequently. The attacking strategy file (Fig. 4) will be updated within every search evolution when it receives mutation proposed attack candidate, to trigger a new fault injection in the designated location for next fitness function evaluation in FPGA. z evolution is needed for one attack iteration.

Fitness Function Evaluation. As discussed above, in a black-box setting, the attacker directly feeds a sample input into the FPGA to evaluate the fitness function in step ③. As the attacker can only access the output prediction from FPGA, he/she can compute the loss function using Eqn.1 and Eqn.2 for un-targeted and targeted attack, respectively. The above process ①-②-③ continues for z evolution times to select one winner candidate to finish one attack iteration. Then, it goes to the next iteration until the attack objective is achieved.

6 Experimental Setup

6.1 Dataset and DNN Models

In our experiment, we evaluate three classes of datasets. First, we use CIFAR-10 [90] and ImageNet [3] for image classification tasks. The other application is object detection where we evaluate the attack on the popular COCO [91] dataset.

For CIFAR-10 dataset, we evaluate the attack against popular ResNet-20 [4] and VGG-11 [92] networks. We use the same pre-trained model with exact configuration as [56, 89]. For ImageNet results, we evaluate our attack performance on MobileNetV2 [93], ResNet-18 and ResNet-50 [4] architectures. For MobileNetV2 and ResNet-18, we directly downloaded a pre-trained model from PyTorch Torchvision models [2] and perform an 8-bit post quantization same as previous attacks [27, 56]. For the ResNet-50, we use Xilinx 8-bit quantized weight trained on ImageNet from [94]. The model we use to validate the YOLOv2 is the official weight [95], trained by COCO [91] dataset, and we quantize [96] each weight value into 16-bits. Our code is also available publicly[3].

6.2 FPGA Prototype Configurations

To validate the real-world performance of Deep-Dup, we develop a multi-tenant FPGA prototype, using a ZCU104 FPGA evaluation kit with an ultra-scale plus family MPSoC chip, which has the same FPGA structure as these used in a commercial cloud server (e.g., AWS F1 instance), running the above

[2] https://pytorch.org/docs/stable/torchvision/models.html
[3] https://github.com/ASU-ESIC-FAN-Lab/DEEPDUPA

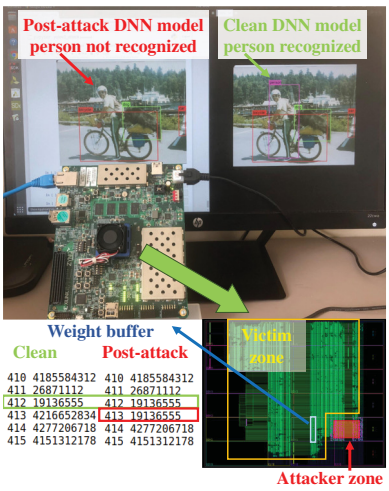


Figure 8: Experimental setup and results of Deep-Dup black-box attack on YOLOv2, with 'person' as target group. After attack, the fault-injected YoLov2 model fails to recognize the 'person'.

discussed deep learning applications: image classification and object detection. The 8-bit quantized DNN models are deployed to our FPGA prototype through a high-level synthesis (HLS) tool, PYNQ frameworks, and CHaiDNN library from Xilinx [94]. The experimental setup is shown in Fig. 8. For object detection (i.e. YOLOv2) FPGA implementation, multiple types of hardware accelerators (HAs) are used to compute different network layers, such as convolution layer, max-pooling layer, and reorganization layer. Specially, the region layer and data cascade are assigned to the ZYNQ's ARM core. For image recognition (e.g. ResNet-50) FPGA implementation, we follow the same design as the Xilinx mapping tool, which only implements the convolution accelerator in a light version (DietChai) [94]. Without loss of generality, the FPGA configurations follow the official parameters [97] and [94]. Object detection network (i.e. YOLOv2) in FPGA execution frequency is 180MHz on Image recognition DNN network (e.g. ResNet-50) in FPGA execute frequency is 150MHz/300MHz, where the DSP uses a 300MHz clock source to increase the throughput and for the other logic we use a 150MHz clock.

To emulate a multi-tenant FPGA environment, we divide the FPGA resources into victim and attacker zones, respectively. The victim zone runs target DNN models, like YOLOv2 or ResNet-50, while the attacker zone mainly consists of malicious power-plundering circuits. Moreover, to limit the available resources of attacker, only 13.38% of the overall FPGA resources are assigned for the power-plundering circuits.

6.3 Evaluation Metric and Hyper-parameters

For classification application, we use *Test Accuracy (TA)* as the evaluation metric. Test Accuracy is the percentage of samples correctly classified by the network. We denote the test accuracy after the attack as *Post-Attack TA*. For a targeted attack, we use *Attack Success Rate (ASR)* to evaluate the performance of the attack; ASR is the percentage of the target class samples miss-classified to an incorrect class after an attack. For the object detection application, we use *Mean Average Precision (mAP)* as the evaluation metric that is the primary metric in the official COCO dataset challenge website[4]. In P-DES, the attack evolution (z) is set to (500/1000) (white-box) and 100 (black-box). In our un-targeted attack, we use a test batch containing 256/25 images for the CIFAR-10/ImageNet dataset. Our code is available publicly[5] with detailed hyper-parameters .

7 Experimental Validation and Results

7.1 Measured Fault Injection Success Rate

As described in Fig. 2, the AWD attack targets the weight transmission procedure, and the fault injection may not always succeed. However, it is infeasible to validate such fault injection success rate in our black-box attack model, in which the adversary has no access to the manipulated weight packages. To measure that, we design another experiment using an AXI4-based weight transmission with the same YOLOv2 setup, i.e., the same memory copy operation. We define the burst length of AXI4 as 256. The entire YOLOv2 int16 quantized weight (99496KB) needs 99496 bursts to finish the transmission for one input image inference. To avoid an FPGA system crash, we only trigger one attack at the middle transmission moment of a burst. To mimic the practical multi-tenant environment with the victim DNN model being executed simultaneously, we run a YOLOv2 in parallel. The available power-plundering circuits are also the same as that in Sec. 6.2. Using this experimental setup, we measured the success rates of fault injection by RO and LRO power-plundering circuits are 84.84% and 58.91%, respectively.

FPGA system crash avoidance. It has been discussed in prior work [80] that a too-aggressive power attack (i.e., leveraging a large power-plundering circuit, or triggering it with unsuitable frequency and duty-cycle) will possibly cause an FPGA system crashes. In our case study, we limit the hardware resources available to the adversary. Additionally, to avoid such system crash, we apply two constraints on the triggering of AWD attacks: 1) A short activation period of each fault injection and 2) A large enough interval between any two consecutive fault injections. Specially, our experiment sets each fault injection period to 50 ns, from which we did not observe a crash of the FPGA setup. The attacking interval between each two consecutive fault injection is set to be longer than 600 ns, which is handled by our P-DES algorithm development, i.e., searching for target attack indexes with a certain distance in between.

[4] https://cocodataset.org/#detection-eval
[5] https://github.com/ASU-ESIC-FAN-Lab/DEEPDUPA

Table 1: Summary of the White-Box Attack on CIFAR-10 and ImageNet Dataset. Here, t_s denotes the target class which we randomly selected for each cases. The attack number is the best number out of three test rounds due to randomness.

White-Box Attack on Image Recognition				Un-Targeted Attack		Targeted Attack			
Dataset	Network	# of Parameters	TA (%)	Post-Attack TA (%)	# of Attacks	Post-Attack TA (%)	Target Class(t_s)	ASR (%)	# of Attacks
CIFAR-10	ResNet-20	0.27 M	90.77	10.92	28	21.63	Bird	99.2	14
	VGG-11	132 M	90.38	10.94	77	23.68	Horse	98.6	63
ImageNet	MobileNetV2	2.1 M	70.79	0.19	1	8.93	Lesser Panda	100.0	1
	ReNet-18	11 M	69.35	0.18	106	34.45	Ostrich	100.0	13
	ReNet-50	23 M	72.97	0.19	175	30.57	Ostrich	100.0	20

Table 2: Black-Box targeted attack results for ImageNet.

Black-Box Targeted Attack on ResNet-50 using RO cell				
(t_s)	TA(%)	Post-Attack TA(%)	ASR (%)	# of Attacks
Ostrich	72.97	46.96	100	26

7.2 White-Box Attack Results

Image Classification Task. We evaluate the proposed Deep-Dup white-box attack framework (in Fig. 6) on two popular Image Classification datasets in Tab. 1. First, for CIFAR-10, our attack achieves close to the target random guess level accuracy (e.g., 10 % for CIFAR-10) with only *28* attack iterations (un-targeted) on ResNet-20. However, to deteriorate the test accuracy of VGG-11 to 10.94 % from 90.38 %, Deep-Dup requires *77* attacks. Similarly, for targeted attack on CIFAR-10, the attacker requires only *14* and *63* attacks to achieve close to *99.0* % ASR on ResNet-20 and VGG-11 respectively. Clearly, VGG-11 is more robust to Deep-Dup attack. We provide the detailed analysis of this phenomenon in sec.8.

For ImageNet dataset, our attack succeeds in degrading the test accuracy of MobileNetV2 to *0.19* % from *70.79* % with just one *single* attack. Even for the targeted attack, it only requires one attack to achieve *100* % ASR in miss-classifying all *Lesser Panda* images. Again, MobileNetV2 is also found to be extremely vulnerable by previous adversarial weight attack [28] as only a single bit memory error can cause catastrophic output performance. Nevertheless, MobileNet is an efficient and compact architecture ideal for mobile and edge computing platforms like FPGA [98]. Thus the vulnerability of these compact architectures against Deep-Dup raises a fair question of how secure are these DNN models in cloud FPGA? The answer from our Deep-Dup attack is a *big NO*. Our attack also succeeds in all ResNet families. Also, larger DNN models (e.g., ResNet-18 & ResNet-50) shows better resistance to Deep-Dup attack.

7.3 Black-Box Attack Results

For proof of concept of our proposed Deep-Dup black-box framework shown in Fig. 7, in this section, we demonstrate and validate the black-box attack on Resnet-50 for image classification task and YOLOv2 for the object detection task.

Table 3: Black-Box attack for object detection.

Black-Box Un-Targeted Attack on YOLOv2 using RO cell			
Target Class (t_s)	mAP	Post- Attack mAP	# of Attacks
All	0.428	0.06	30
Black-Box Un-Targeted Attack on YOLOv2 using LRO cell			
Target Class (t_s)	mAP	Post- Attack mAP	# of Attacks
All	0.428	0.14	63
Black-Box Targeted Attack on YOLOv2 using RO cell			
Target Class (t_s)	AP	Post-Attack AP	# of Attacks
Person	0.6039	0.0507	20
Car	0.5108	0.0621	18
Bowl	0.3290	0.0348	15
Sandwich	0.4063	0.0125	6

Specially, in our case study, we randomly pick the "ostrich" class in the Imagnet dataset as a target class for ResNet-50 and 4 target objects (i.e. Person, Car, Bowl and Sandwich) in the COCO dataset for YOLOv2. Other settings and performance metrics are the same as described in Sec. 7.2. Note that, all the black-box results are the actual measurement from our FPGA prototype. The Deep-Dup black-box attack on ResNet-50 are successful and results are reported in Tab. 2. It can be seen that only 26 attacks are needed to attack the “ostrich” with 100 % ASR. Similarly, Deep-Dup black-box un-targeted and targeted attacks on YOLOv2, with both RO and LRO cells, are also successful, as reported in Tab. 3. It can be seen that the post-attack average precision (AP) is significantly degraded after less than 20 attacks. For example, only 6 attacks are needed to decrease the AP of sandwich class from 0.4063 to 0.0125.

7.4 Comparison to Other Methods

Previously, very few adversarial weight attack works have been successful in attacking DNN model parameters to cause complete malfunction at the output [26, 29]. Thus we only compare with the most recent and successful adversarial bit-flip (BFA) based weight attack [27, 28], which uses a gradient-based search algorithm to degrade DNN performance in a white-box setting. We also compare our search algorithm (P-DES) to a random AWD attack.

Table 4: Comparison of Deep-Dup with *random* AWD attack and row-hammer based (*BFA* [27, 28]) attack. All the results are presented for 8-bit quantized VGG-11 model [27].

Method	Threat Model	TA (%)	Post-Attack TA (%)	# of Attacks
Random	Black Box	90.23	90.04	100
BFA [28]	White Box	90.23	10.8	28
Deep-Dup	Black & White Box	90.23	10.94	77

As shown in both Tab. 4 , only *77* AWD attack iterations can degrade the accuracy of VGG-11 to *10.87 %* while randomly performing *100* AWD attacks, cannot even degrade the model accuracy beyond *90 %*. On the other hand, a BFA attack [28] using row-hammer based memory fault injection technique, requires only 28 attacks (i.e. memory bit-flips) to achieve the same un-targeted attack success (i.e., $\sim$ 10 % TA). However, BFA attack is only successful for white-box setting, not black-box.

7.5 Discussion

Attack efficiency w.r.t. fault injection success rate. As described in section 7.1, we used two different power plundering circuits, i.e., RO and LRO for fault injection. In our experiments, we measured 84.84% and 58.91% fault injection success rates for RO and LRO, respectively. In practical attack, this number may vary due to the attack budget (i.e., frequency, resource, etc.). In order to validate our Deep-Dup attack framework will succeed in different fault injection success rates, we incorporate the fault success rate as a probabilistic parameter in our off-line simulator as discussed in section 5.3.1. Note that, for black-box attack, our direct evaluation of fitness function in the FPGA accelerator already considers and compensates for the failed fault iteration. The experimental results are shown in Tab.5. We observe that our Deep-Dup attack framework could still succeed at very low fault injection success rate (i.e., 40 %), but requiring more number of attack iterations (i.e. higher redundancy as explained in sec.5.3).

Table 5: Attack efficiency v.s. fault injection success rate (f_p). Reporting # of attack iterations (i.e., mean $\pm$ std. for three runs) required to achieve 99.0 % ASR (targeted attack) or 11.0 % test accuracy (un-targeted attack).

Model	Type	40 %	60 %	80 %
ResNet-20	Un-Targeted	95.3 ± 37.3	88 ± 66.5	76.6 ± 13.8
	Targeted	39 ± 7.8	23.3 ± 4.3	23.8 ± 6.8
VGG-11	Un-Targeted	195.3 ± 39.1	95.6 ± 14.1	98.9 ± 1.9
	Targeted	114 ± 32	88.6 ± 34.4	62.6 ± 2.6

Attack Time Cost. The execution time of one searching iteration of our proposed P-DES algorithm is constant for a fixed z, regardless of DNN model size. The overall searching time is proportional to the number of evolution (z). For Deep-Dup white-box attack, the P-DES algorithm is executed offline, and the AWD attack is only executed when the attack index is generated. Note that, the hardware AWD attack incurs no time cost, as it runs in parallel with the victim DNN model. For Deep-Dup black-box attack, two main time cost includes mutation generation (proportional to z) and FPGA fitness function evaluation (proportional to DNN acceleration performance/latency in FPGA). In Fig. 9, we report the average time cost of the proposed 4 mutation strategies executed in the PS of our FPGA prototype. Additionally, we also report the DNN execution time in FPGA, which is determined by the corresponding DNN model size, architecture, optimization method, and available FPGA hardware resources. It is easy to observe that our P-DES mutation generation only consumes trivial time compared to DNN execution time in FPGA, which is the bottleneck in black-box attack.

Task	Network	Model quantization	Training set	Mutation generate time (ms)	FPGA acceleration time (ms/image)
Classification	ResNet-50	8-bits	ImageNet	16.0175	588
Object detection	YOLO-V2	16-bits	COCO	15.075	914

Figure 9: Black-Box attack time cost analysis with $z = 100$. FPGA acceleration (i.e., fitness function evaluation) time and mutation generation time are reported.

8 Potential Defense Analysis

Increasing Model Redundancy. Several prior works have demonstrated that increasing model redundancy (i.e., DNN size/channel width) [89, 99] can be a potential defense against model fault attack. Our evaluation of Deep-Dup attack in the previous section also indicates the correlation between network capacity (i.e., # of model parameters) and model robustness (# of attacks required). As the ImageNet dataset section depicts in Tab. 1, as the network size increases from ResNet-18 to ResNet-50, the number of attacks required to achieve 100 % ASR increases correspondingly. We observe the same trend for CIFAR-10 models where VGG-11 (i.e., dense model) requires a higher number of attacks than ResNet-20 (i.e., compact model).

Table 6: Attack efficiency after increasing the model size of ResNet-20 and VGG-11 model by 4 (i.e., increasing each input and output channel size by 2).

Method	ASR(%)	# of Attacks
ResNet-20 (*Baseline*)	99.6	14
ResNet-20 $\times$ *4*	99.6	*21*
VGG-11 (*Baseline*)	98.6	63
VGG-11 $\times$ *4*	98.2	*84*

In Tab. 6, we run an experiment to validate the relation between Deep-Dup attack efficiency and network model size. First, we multiply the input and output channel of the baseline model by 2 to generate ResNet-20 ($\times$ 4) and VGG-11 ($\times$ 4) models with 4 $\times$ larger capacity. For both ResNet-20 and VGG-11, the number of attacks required to achieve similar ASR increases with increasing model capacity (Tab. 6). To conclude, one possible direction to improve the DNN model's resistance to the Deep-Dup attack is to use a dense model with a larger redundancy.

Protecting Critical Layers. Another possible defense direction is to protect the critical layers that are more sensi-

tive. Prior works [100] have proposed selective hardening to defend against weight faults by selectively protecting more sensitive layers. It is interesting to note that our experimental observation also shows that 80 % of the searched vulnerable weights are within the first two layers and the last layer for ResNet-20. Following this observation, in Tab. 7, we run our attack by securing these three sensitive layers (*ResNet-20 (Protected)*). A straightforward way to secure layer weights from Deep-Dup would be to store them on-chip (i.e., no need for off-chip data transfer). Note that, a defender can not store an entire DNN model on-chip due to limited on-chip memory and typically large DNN model size for cloud computing. Nevertheless, as shown in Tab. 7, our Deep-Dup still manages to succeed with $\sim 2\times$ additional rounds of attack on the protected ResNet-20 model. Similarly for VGG-11, our Deep-Dup attack still successfully achieves $\sim$ 99.0 % ASR even after securing some critical DNN layers from fault attacks.

Table 7: Deep-Dup attack performance after protecting or securing some critical DNN layers

Method	ASR(%)	# of Attacks
ResNet-20 (*Baseline*)	99.6	14
ResNet-20 (*Protected*)	99.2	*29*
VGG-11 (*Baseline*)	98.6	63
VGG-11(*Protected*)	98.2	141

Obfuscation through Weight Package Randomization. In our Deep-Dup attack, the P-DES algorithm relies on the sequence (e.g., index) of the weight packages being transferred between the on-chip buffer and off-chip memory. In this section, we discuss the possibility of defending our attack by introducing random weight package transmission as an obfuscation scheme. In Tab. 8, we first perform an experiment with shuffling of the weights in a pre-defined sequence before transmitting them. The results show that pre-defined shuffling order of the wights has almost no effect on the attack efficacy.

Table 8: Weight package randomization as obfuscation. **Pre-defined Shuffle** : Shuffling the weight packages in a pre-defined order before transmission. **Random Shuffle** : Shuffling the weight packages every time using a random function before transmission.

Method	TA (%)	Post-Attack TA (%)	# of Attacks
Random Attack	90.77	87.9	180
ResNet-20 Baseline	90.77	10.94	28
Pre-defined Shuffle	90.77	11.0	26
Random Shuffle	90.77	53.3	180

Next, we discuss the case with shuffling the weight package for every transmission round as a very strong obfuscation. The effect of such a strong obfuscation scheme can have three possible implications. First, a randomly shuffled weight transmission will fail to defend our attack in a white-box setting as the attacker has full knowledge of the DNN and data transmission scheme. Second, in a black-box setting, as shown in Tab. 8, this defense will greatly limit the efficacy of our attack, requiring a larger amount of attack iterations (e.g., 180) to degrade the accuracy to 53.3 %. But the attack remains more successful than a random AWD attack with no searching algorithm. It aligns with the recent work of adversarial input attack [23], where the authors argue that obfuscation based on an under-lying random function as defense may not completely defend a progressive adversarial attack. Given a large amount of model query, the progressive evolutionary algorithm-based attack (i.e. our case) could estimate the effect and distribution of the randomness to improve the attack efficacy in comparison to a random attack. Moreover, randomly shuffling data transmission every time would require additional header information to synchronize the sequence of weights at the receiver end. A recent work in [101] has demonstrated random shuffling may cost up to 9 $\times$ energy in-efficiency and 3.7 $\times$ lesser amount of throughput. Thus, an effective defense scheme will always come at the expense of additional (i.e., memory, speed & power) overhead.

Power-based side-channel analysis to detect Deep-Dup. Here we discuss the feasibility of using power-based side-channel analysis to detect Deep-Dup. The success of such detection should rely on the ability to distinguish between these two cases: 1) *Normal case*: two benign users execute their applications simultaneously, and 2) *Attack case*: two users share the FPGA resources, where one of them apply Deep-Dup to attack the other one. Since it is impractical to measure the real-time power trace in a cloud-FPGA with an oscilloscope, an on-chip power sensor (e.g., TDC sensor) will be the only option. As shown in Fig.4, similar as AWD attack, our measured power trace of a benign user (e.g., YOLOv2) also incurs large power glitches. More importantly, we did not observe any AWD attack power glitch has a larger magnitude than that of benign user-YOLOV2. Instead, it is smaller for most of the time. Therefore, the glitches caused by AWD will be easily obfuscated. Further, it is difficult to distinguish AWD power glitches in the following practical scenarios: i) Most cloud-FPGA users prefer to run compute-intensive applications, which generates many power glitches; ii) When triggered, each fault injection by AWD only lasts for a short time period (e.g., 50ns) and is disabled for most of the time; iii) Faults are only injected at attacker's will, i.e., without a fixed pattern to check. In other words, it is of different challenges to use such power-based side-channel analysis for defense and attack, i.e., the defender should acquire ultra-high-resolution side-channel information to identify the malicious power glitches from the noisy power background by the compute-intensive application, e.g., the DNN execution; while the attacker only needs to identify the temporal range for the DNN weight transmission. More severely, an attacker may even choose to inject faults in a more stealthy manner, i.e., while the victim DNN model itself is generating lots of power glitches, to exacerbate the overall voltage drop [102]. Therefore, we argue that it is extremely difficult, if not impos-

sible, to detect the proposed Deep-Dup attacks with power anomaly in a multi-tenant FPGA.

9 Conclusion

In this work, we study the security of DNN acceleration in multi-tenant FPGA. For the first time, we exploit this novel attack surface where the victim and the attacker share the same FPGA hardware sources. Our proposed Deep-Dup attack framework is validated with a multi-tenant FPGA prototype, as well as some popular DNN architectures and datasets. The experimental results demonstrate that the proposed attack framework can completely deplete DNN inference performance to as low as random guess or attack a specific target class of inputs. It is worth mentioning that our attack succeeds even assuming the attacker has no knowledge about the DNN inference running in FPGA, i.e. black-box attack. A malicious tenant with such limited knowledge can implement both targeted and un-targeted malicious objectives to cause havoc for a victim user. Finally, we envision that the proposed attack and defense methodologies will bring more awareness to the security of deep learning applications in the modern cloud-FPGA platforms.

Acknowledgement: The authors thank the designated shepherd (Dr. Nele Mentens) for her guidance, and the anonymous reviewers for their valuable feedback. This work is supported in part by the National Science Foundation under Grant No.2019548 and No.2043183.

References

[1] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara N Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[7] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[8] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi. Deep ehr: A survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE Journal of Biomedical and Health Informatics*, 22(5):1589–1604, Sep. 2018.

[9] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: Deep learning in android malware detection. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 371–372. ACM, 2014.

[10] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 2722–2730. IEEE, 2015.

[11] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla, and R. Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1013–1020, June 2018.

[12] Altera and ibm unveil fpga-accelerated power systems. https://www.hpcwire.com/off-the-wire/altera-ibm-unveil-fpga-accelerated-power-systems/.

[13] Here's what an intel broadwell xeon with a built-in fpga looks like, 2016. https://www.theregister.co.uk/2016/03/14/intel_xeon_fpga/.

[14] Inside the microsoft fpga-based configurable cloud, 2017. https://azure.microsoft.com/en-us/resources/videos/build-2017-inside-the-microsoft-fpga-based-configurable-cloud/.

[15] Enable faster fpga accelerator development and deployment in the cloud, 2020. https://aws.amazon.com/ec2/instance-types/f1/.

[16] George Provelengios, Daniel Holcomb, and Russell Tessier. Power wasting circuits for cloud fpga attacks. In *30th International Conference on Field Programmable Logic and Applications (FPL)*, 2020.

[17] Yue Zha and Jing Li. Virtualizing fpgas in the cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 845–858, 2020.

[18] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[19] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[20] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[22] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[23] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[24] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[25] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[26] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 497–514, 2019.

[27] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[28] Fan Yao, Adnan Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networksthrough targeted chain of bit flips. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[29] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138. IEEE, 2017.

[30] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *arXiv preprint arXiv:2007.12336*, 2020.

[31] Jason Cong, Zhenman Fang, Muhuan Huang, Peng Wei, Di Wu, and Cody Hao Yu. Customizable computing—from single chip to datacenters. *Proceedings of the IEEE*, 107(1):185–203, 2018.

[32] Xilinx: Socs, mpsocs and rfsocs, 2020. https://www.xilinx.com/products/silicon-devices/soc.html.

[33] Intel: Soc fpgas, 2020. https://www.intel.com/content/www/us/en/products/programmable/soc.html.

[34] Mark Zhao and G Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 229–244. IEEE, 2018.

[35] Jonas Krautter, Dennis RE Gnad, and Mehdi B Tahoori. Fpgahammer: remote voltage fault attacks on shared fpgas, suitable for dfa on aes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 44–68, 2018.

[36] Ilias Giechaskiel, Kasper B Rasmussen, and Ken Eguro. Leaky wires: Information leakage and covert communication between fpga long wires. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 15–27. ACM, 2018.

[37] Yukui Luo and Xiaolin Xu. Hill: A hardware isolation framework against information leakage on multi-tenant fpga long-wires. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 331–334. IEEE, 2019.

[38] Dina Mahmoud and Mirjana Stojilović. Timing violation induced faults in multi-tenant fpgas. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1745–1750. IEEE, 2019.

[39] George Provelengios, Chethan Ramesh, Shivukumar B Patil, Ken Eguro, Russell Tessier, and Daniel Holcomb. Characterization of long wire data leakage in deep submicron fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 292–297. ACM, 2019.

[40] Machine learning on aws, 2020. https://aws.amazon.com/machine-learning/?nc1=h_ls.

[41] Cloud automl, 2020. https://cloud.google.com/automl.

[42] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. *CoRR*, 2017.

[43] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of International Conference on Machine Learning, ICML 2018*, July 2018.

[44] Ekin D Cubuk, Barret Zoph, Samuel S Schoenholz, and Quoc V Le. Intriguing properties of adversarial examples. *ICLR workshop*, 2018.

[45] Giuseppe Ateniese, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *Int. J. Secur. Netw.*, 10(3):137–150, September 2015.

[46] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 601–618. USENIX Association, 2016.

[47] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 506–519. ACM, 2017.

[48] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333. ACM, 2015.

[49] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *USENIX Security Symposium*, pages 17–32, 2014.

[50] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *arXiv preprint arXiv:1612.06299*, 2016.

[51] Battista Biggio, Luca Didaci, Giorgio Fumera, and Fabio Roli. Poisoning attacks to compromise face templates. In *Biometrics (ICB), 2013 International Conference on*, pages 1–7. IEEE, 2013.

[52] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[53] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy*, pages 3–18, May 2017.

[54] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model extraction warning in mlaas paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 371–380. ACM, 2018.

[55] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Deeplaser: Practical fault attack on deep neural networks. *arXiv preprint arXiv:1806.05859*, 2018.

[56] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13198–13207, 2020.

[57] Chethan Ramesh, Shivukumar B Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. Fpga side channel attacks without physical access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[58] Sadegh Yazdanshenas and Vaughn Betz. The costs of confidentiality in virtualized fpgas. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[59] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *13th {USENIX}*

Symposium on Operating Systems Design and Implementation ({OSDI} 18), pages 107–127, 2018.

[60] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 515–532, 2019.

[61] Shayan Moini, Shanquan Tian, Jakub Szefer, Daniel Holcomb, and Russell Tessier. Remote power side-channel attacks on cnn accelerators in fpgas. *arXiv preprint arXiv:2011.07603*, 2020.

[62] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020.

[63] Oliver Knodel, Patrick Lehmann, and Rainer G Spallek. Rc3e: Reconfigurable accelerators in data centres and their provision by adapted service models. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 19–26. IEEE, 2016.

[64] Sadegh Yazdanshenas. *Datacenter-optimized FPGAs*. PhD thesis, 2019.

[65] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–597, 2019.

[66] Adnan Siraj Rakin, Zhezhi He, Li Yang, Yanzhi Wang, Liqiang Wang, and Deliang Fan. Robust sparse regularization: Defending adversarial attacks via regularized sparse network. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 125–130, 2020.

[67] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

[68] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[69] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017.

[70] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.

[71] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnexplorer: A framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator. *arXiv preprint arXiv:2008.12745*, 2020.

[72] Pengfei Xu, Xiaofan Zhang, Cong Hao, Yang Zhao, Yongan Zhang, Yue Wang, Chaojian Li, Zetong Guan, Deming Chen, and Yingyan Lin. Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 40–50, 2020.

[73] ARM. *AMBA AXI and ACE Protocol Specification*, 2013.

[74] Xilinx,Inc. *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS181)*, 2018.

[75] Xilinx,Inc. *Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics (DS183)*, 2019.

[76] Xilinx,Inc. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics (DS925)*, 2019.

[77] Power distribution network, 2015. https://www.intel.com/content/www/us/en/programmable/support/support-resources/support-centers/signal-power-integrity/power-distribution-network.html.

[78] TI,Inc. *TPS54620 4.5-V to 17-V Input, 6-A, Synchronous, Step-Down SWIFT™ Converter*, 2017.

[79] Xilinx,Inc. *UltraScale Architecture PCB Design (UG583)*, 2020.

[80] Dennis RE Gnad, Fabian Oboril, and Mehdi B Tahoori. Voltage drop-based fault attacks on fpgas using valid bitstreams. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[81] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. Fpgadefender: Malicious self-oscillator scanning for xilinx ultrascale+

fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 13(3):1–31, 2020.

[82] Kaspar Matas, Tuan Minh La, Khoa Dang Pham, and Dirk Koch. Power-hammering through glitch amplification–attacks and mitigation. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 65–69. IEEE, 2020.

[83] Yukui Luo, Cheng Gongye, Yunsi Fei, and Xiaolin Xu. Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga. *arXiv preprint arXiv:2105.09453*, 2021.

[84] David G Mayer, BP Kinghorn, and Ainsley A Archer. Differential evolution–an easy and efficient evolutionary algorithm for model optimisation. *Agricultural Systems*, 83(3):315–328, 2005.

[85] Kenneth V Price. Differential evolution. In *Handbook of Optimization*, pages 187–214. Springer, 2013.

[86] Libiao Zhang, Xiangli Xu, Chunguang Zhou, Ming Ma, and Zhezhou Yu. An improved differential evolution algorithm for optimization problems. In *Advances in Computer Science, Intelligent System and Environment*, pages 233–238. Springer, 2011.

[87] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. Recent advances in differential evolution–an updated survey. *Swarm and Evolutionary Computation*, 27:1–30, 2016.

[88] Feoktistov Vitaliy. Differential evolution–in search of solutions, 2006.

[89] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14095–14103, 2020.

[90] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

[91] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[92] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[93] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[94] Chaidnn, hls based deep neural network accelerator library for xilinx ultrascale+ mpsocs. https://github.com/Xilinx/CHaiDNN, 2018.

[95] Yolo-v2 pre-trained weight. https://pjreddie.com/media/files/yolov2.weights, 2016.

[96] Lei Shan, Minxuan Zhang, Lin Deng, and Guohui Gong. A dynamic multi-precision fixed-point data quantization strategy for convolutional neural network. In *CCF National Conference on Computer Engineering and Technology*, pages 102–111. Springer, 2016.

[97] Yolov2 accelerator in xilinx's zynq-7000 soc. https://github.com/dhm2013724/yolov2_xilinx_fpga.

[98] Di Wu, Yu Zhang, Xijie Jia, Lu Tian, Tianping Li, Lingzhi Sui, Dongliang Xie, and Yi Shan. A high-performance cnn processor based on fpga for mobilenets. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 136–143. IEEE, 2019.

[99] Yu Li, Yannan Liu, Min Li, Ye Tian, Bo Luo, and Qiang Xu. D2nn: a fine-grained dual modular redundancy framework for deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 138–147, 2019.

[100] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech. Selective hardening for neural networks in fpgas. *IEEE Transactions on Nuclear Science*, 66(1):216–222, 2019.

[101] Shijie Zhou, Charalampos Chelmis, and Viktor K Prasanna. High-throughput and energy-efficient graph processing on fpga. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 103–110. IEEE, 2016.

[102] Yukui Luo, Cheng Gongye, Shaolei Ren, Yunsi Fei, and Xiaolin Xu. Stealthy-shutdown: Practical remote power attacks in multi-tenant fpgas. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 545–552. IEEE, 2020.