UNIVERSITY OF CALIFORNIA,
IRVINE

Stealing Deep Learning Model Secret through Remote FPGA Side-channel Analysis

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Yicheng Zhang

Thesis Committee:
Associate Professor Mohammad Abdullah Al Faruque, Chair
Assistant Professor Zhou Li
Assistant Professor Yanning Shen

2021

DEDICATION

To my parents,
Junxiong Zhang and Fengju Chen,
for always loving and supporting me,
and to Jinrun Hu,
my lovely wife,
who has been a constant source of support and encouragement from the day we met.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my committee chair, Professor Mohammad Abdullah Al Faruque, for constantly supporting me and advising me.

I would also like to thank my committee members, Professor Zhou Li, and Professor Yanning Shen, for providing me guidance and feedback. Without their help, I can not finish this thesis.

I would also like to thank my lab-mates Rozhin Yasaei and Hao Chen for their great work with exploring power side-channel analysis on the FPGA board and for their intellectual support with my research.

ABSTRACT OF THE THESIS

Stealing Deep Learning Model Secret through Remote FPGA Side-channel Analysis

By

Yicheng Zhang

Master of Science in Computer Engineering

University of California, Irvine, 2021

Associate Professor Mohammad Abdullah Al Faruque, Chair

Companies have extensively developed deep Neural Network (DNN) models for a wide range of applications. The development of a customized DNN model with great performance requires costly investments, and its structure (layers and hyper-parameters) is considered intellectual property and holds immense value. However, in this paper, we found the model secret is vulnerable when a cloud-based FPGA accelerator executes it. We demonstrate an end-to-end attack based on remote power side-channel analysis and machine-learning-based secret inference against different DNN models. The evaluation result shows that an attacker can reconstruct the layer and hyper-parameter sequence at over 90% accuracy using our method, significantly reducing their model development workloads. We believe the threat presented by our attack is tangible, and new defense mechanisms should be developed against this threat.

Chapter 1

Introduction

Recently machine-learning techniques, especially Deep Neural Networks (DNN), have attracted substantial attention due to their prominent performance in solving complex problems like image recognition and natural language processing. The development of a customized machine learning model requires companies to invest ample computation and human resources. Therefore, the developed model has high values as an intellectual property [36], which also makes it a target for attackers trying to violate its *confidentiality*.

The composition of layers and hyper-parameters distinguishes DNN models. When the combination of a layer and its hyper-parameters result in unique hardware traits, inferring the model secret could be possible through *side-channel attack*. It has been demonstrated that stealing model secret is feasible when launching CPU [60, 12, 18], GPU [33, 19, 55, 69] side-channel attacks on the *conventional* shared computing platforms. On the other hand, whether the emerging cloud platform powered by *FPGA* is vulnerable to such a model-stealing attack has never been studied before.

We believe that exploring the attack surface on FPGA is very important and needs urgent attention. Major cloud providers began to provide FPGA-based instances, like Amazon AWS

F1[2], Google Compute Engine (GCE)[16], and Microsoft Azure[30], to accelerate DNN training and testing, in part of the better energy utilization of FPGA. To allow flexible resource allocation of the FPGA instances, FPGA virtualization is proposed to allow multiple users to deploy their logic on the same FPGA board and run concurrently [52]. Unfortunately, security implications could surface under this *multi-tenant FPGA* scenario. Though FPGA virtualization ensures logic is placed into slots separated physically and logically, there is no guarantee that one logic's execution status is completely opaque to others.

Motivated by previous attacks targeting the multi-tenant FPGA [68], which aims to steal the encryption keys, in this paper, we carry out the remote power side-channel analysis on the shared FPGA instance to evaluate the feasibility of model-stealing attacks. We developed an attack method that can steal model secret *remotely* without any physical access to FPGA instance through measuring the *power consumption*.

Specifically, we found when the attacker deploys a power sensor like ring oscillators on the same FPGA board where the victim's DNN model is executed, by leveraging machine-learning-based inference models (e.g., XGBoost [7]), both layers and their hyper-parameters can be distinguished. We implemented our attack based on this observation on a ZedBoard [58] and evaluated end-to-end attacks on three DNN models: multiple layer perceptron(MLP) [26], AlexNet [24], and VGG16 [47]. Our evaluation result shows that the attack can correctly infer over 90% secret elements (layers and hyper-parameters) for all three models, and 94.52% accuracy can be achieved on VGG16, which is still widely used today. To notice, our attack can succeed *even when the models used for attack training and testing are from two families*, suggesting the generality of our attack method. As such, we conclude that the threat of an FPGA-based model-stealing attack is practical, and new defense solutions should be developed. Below we highlight the research challenges faced in our work and our technical contributions.

Research challenges. Though there have been attacks presented which breach the confi-

dentiality of crypto cores or DNN models, they have shortcomings to address the following challenges in real cloud scenario:

- Lacking physical access to the victim hardware platform makes it impossible to apply local side-channel attacks [20, 4, 56, 62].
- The input and output data to the targeted model is not available to the adversary. Thus, conventional power analysis methods such as SPA [27] and DPA [68] would fail.
- The confidentiality of DNN depends on a broad set of elements (hyper-parameters and layers), which contains huge search space while crypto cores only have only one secret encryption key.

Contributions. This paper presents the **first** attack exploiting remote power side-channel for DNN model stealing and addresses the aforementioned challenges to the best of our knowledge. The followings are our contributions:

- The first attack to reconstruct the DNN model's architecture without physical access to the FPGA platform.
- Trained inference models to infer DNN secret, including layers and hyper-parameters.
- An end-to-end evaluation of popular DNN models like MLP, AlexNet, and VGG.

Chapter 2

Background and Related Work

In this section, we first illustrate how the FPGA instances are organized on the cloud. We then describe the setting and goal of the adversary. Finally, we overview related work to our attack.

2.1 Cloud FPGAs

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated silicon devices that can be electrically configured to become nearly any kind of digital circuit or system. Compared to the fixed-function Application Specific Integrated Circuit (ASIC), FPGAs can be programmed in less than a second and cost much less. Thus, FPGAs have been widely adopted in large-scale data centers to improve computing systems' performance and energy efficiency. In cloud computing, performance and power consumption are two of the most notable challenges. To address these two challenges, FPGAs have been deployed due to their flexible programmability and power efficiency. Even though cloud FPGAs providers are still primarily single-tenant, the expanding demand for higher hardware resource utilization already

led to the growth of techniques and architectures supporting *FPGA multi-tenancy*. The main strategy of *FPGA multi-tenancy* is based on *FPGA virtualization* [52] which supports space-sharing of FPGA instances among multiple tenants in the cloud.

Some prior works [49, 10, 66, 1] present methodologies to the temporary allocation of FPGA instances by successively allocating entire or part of FPGA instance to tenants over time. Some research [6, 5, 54, 28, 29] demonstrated cloud platforms enabling spatial FPGA sharing by exposing FPGA regions labeled "virtual FPGAs" to cloud tenants. For instance, some architectures divide each physical FPGA instance into several slots allocated to *virtual instances* (VI). Regarding the security issue of *FPGA multi-tenancy*, logical and physical isolation is provided to safeguard the inter-influence between the *virtual instances* (VI) of cloud tenants. However, none of these methods mitigate the threat from *side-channel leaks*, e.g., under the shared power supply.

Specifically, this paper investigates how remote power side-channel analysis can reveal DNN model structures' secret in such *FPGA multi-tenancy* scenario. Below we describe the setting and goal of the adversary.

2.2 Adversary Model

We assume a victim cloud tenant uses the FPGA instance provided by a public cloud to train or test her DNN model. We assume FPGA is virtualized and multi-tenancy is allowed so that FPGA is partitioned into slots (or VI), and two users can share an FPGA board by programming their own VIs. All VIs are isolated physically and logically. Also, We assume an attacker user can co-locate with the targeted victim and attempts to steal the victim's DNN model secret covertly by programming an adversarial FPGA logic (called spy). By exploiting the side-channel leakage, the spy aims to bypass the VI isolation. To this end, attackers first

rent a VM on the same cloud platform as a tenant. Then, they program an adversarial FPGA logic (called spy) and attempts to deploy it on the same FPGA instance with the victim and carry out side-channel analysis to bypass isolation and infer the confidential information of the victim. To notice, prior work shows adversarial co-location [43, 67] is feasible, and Zhao et al. [68] have demonstrated their attacks against multi-tenant FPGA under the same condition.

To evaluate our attack, we deployed the attack and victim logic on a Xilinx Zynq-7000 FPGA board, Zedboard [58], which is also used by the prior FPGA remote attack [68]. Cloud providers utilize the FPGA board to serve FPGA users, like Xilinx Virtex Ultrascale+ used by Amazon F1 [2], so our attack is expected to apply to the cloud FPGA. We did not test our attack on the real FPGA cloud instance because we have not found a cloud providing multi-tenant FPGA service now (though it is expected to happen in the near future [29]), and it is difficult to run controlled attacks on production cloud without affecting other legitimate users. We discuss this limitation in Section 5. Figure 2.1 illustrates the adversary model.

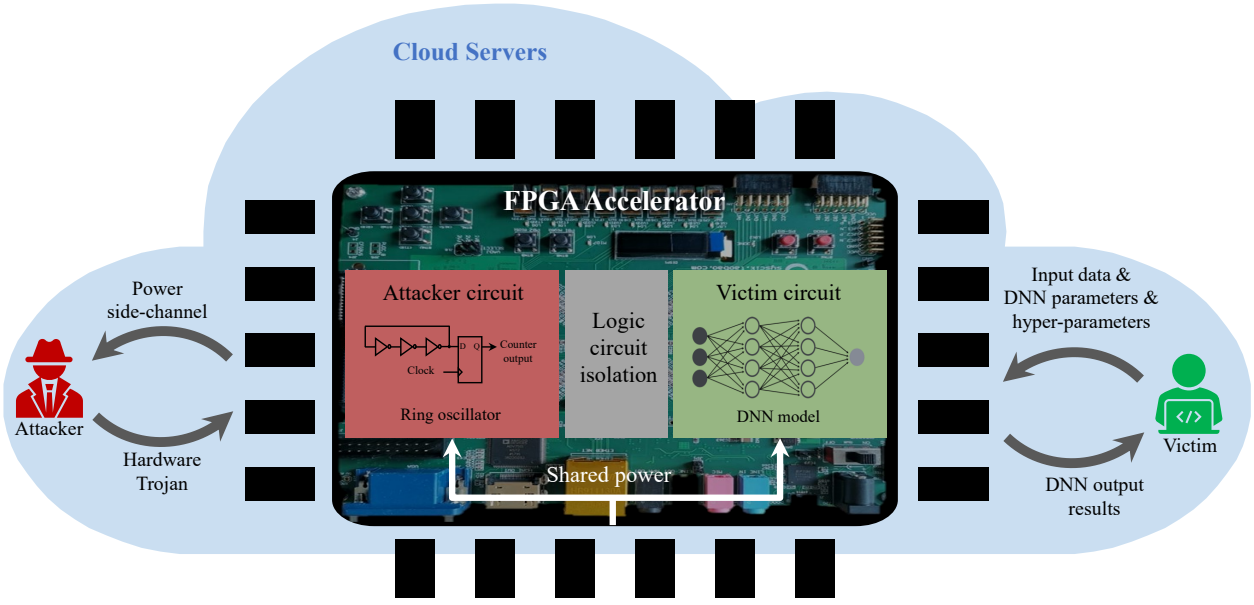

Figure 2.1: Adversary Model

Before our work, Hua et al. carried out the model-stealing attack on FPGA [20]. Still,

they assume the attacker has complete control of the FPGA board, which can 1) feed any input and measure its output and 2) sniff the data bus between FPGA and the off-the-chip memory to learn the fine-grained data access patterns. **In our remote FPGA attack, the attacker can only measure the victim DNN execution *passively* (no control of input) and the leaked information is *coarse-grained* (no data access patterns).** Inferring the DNN structure is much more challenging here. Still, through our novel power-analysis methods, we demonstrate the attack is completely feasible.

2.3 Deep Learning Model Structural Secret

DNN structure has been considered intellectual property for an AI company [36], due to the high human and resource expense involved in developing a DNN model. Stealing DNN structural secret can be seen as finding a solution in a search space that yields sufficient accuracy on the tested data. The search space is vast due to the complicated structure and hyper-parameter settings of DNN. For instance, as analyzed by a prior work [60], the search space for VGG16, a broadly used DNN model, is 5.4×10^{12}, if the attacker guesses in a brute-force way. Figure 2.2 shows the architecture of AlexNet [24], which contains 5 convolutional layers and 3 fully-connected layers.

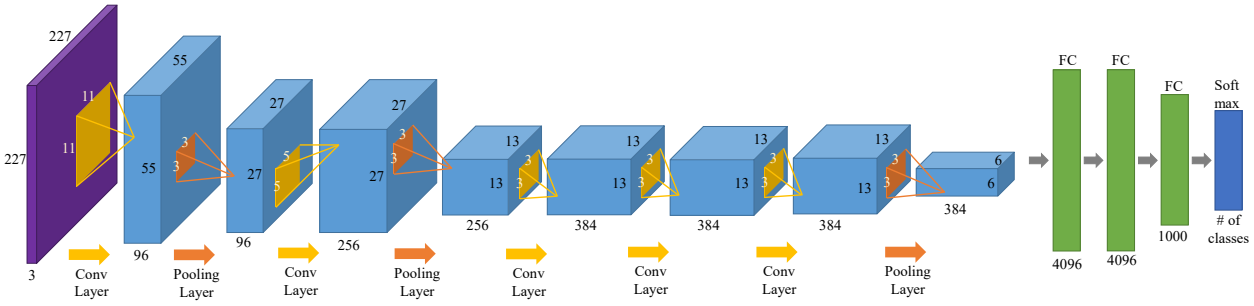

Figure 2.2: The Architecture of AlexNet

Similar to prior works [20, 4, 60, 62, 55, 33], we investigate the model-stealing attack on CNN (Convolutional Neural Network) models and MLP (Multilayer Perceptron) and consider the

structural elements described below as secret.

- The number of layers.
- The type of each layer.
- The activation function used by each layer.
- The number of neurons of each fully-connected layer.
- The filter size of each convolutional layer and pooling layer.
- The number of filters of each convolutional layer.
- The stride of each convolutional layer and pooling layer.
- The connection between layers, sequential or non-sequential.

Our attack can infer **nearly all** model structural secrets, except non-sequential layer connection, which we leave as future work. On the other hand, our attack does not infer the neuron weights because we assume a *passive* adversary. Some prior works carried out *active* analysis [19, 4, 60], feeding arbitrary input and using the change of output to infer the weights. Unfortunately, our setting does not allow input manipulation. We discuss this limitation in Section 5.

2.4 Related Work

We organize the discussion of related work into two different groups: (1) Deep learning model stealing attacks; and (2) Remote FPGA attacks.

2.4.1 Deep Learning Model Stealing Attacks

As the machine-learning model becomes intellectual property, A number of prior works [51, 53, 63, 21, 60, 33, 19, 4, 20, 12, 18, 62, 55, 69, 63, 64, 61] have investigated how to steal the model secret by exploiting the *algorithm* or *hardware* weakness. In the first direction, prior works showed that by issuing queries using the model's public API and analyze the response (e.g., similarity score), it is possible to learn model parameters (e.g., weights of a linear regression model) [51, 53, 63, 64]. Recently, a few works examined the API-based model stealing on DNN models and it turns out when leveraging adversarial examples [63] or functionally-equivalent extraction [21]. In the second direction, prior works demonstrated the possibility to use cache side-channel analysis, EM analysis, and power analysis on CPU [60, 12, 18], GPU [33, 19, 55, 69], microcontroller [4] and FPGA [20, 62, 61] to infer the model secret.

The research closest to ours was done by Hua et al. [20], and Yu et al. [62]. They assume the attacker has complete control of and physical access to the FPGA board, who can 1) feed any input and measure its output, and 2) sniff the data bus between FPGA and the off-the-chip memory or EM side-channel measurements collected by EM probe. However, ours differ in that we investigate the remote FPGA side-channel attack, which more practical and general on the cloud FPGA: **the attacker can only measure the victim DNN execution *passively* (no control of input) and *covertly* (no physical access to FPGA instance)**.

2.4.2 Remote FPGA Attacks

Our attack can be categorized as a specific type of remote FPGA attack [14, 15, 40, 68, 45, 50, 31] as well. The major security concern in this setting is that by programming ring oscillators on FPGA, an adversary can directly access the hardware of victim vendors without physical

access and bypass security measures like hypervisor isolation. ROs and TDCs (Time-to-Digital Converters) are powerful as voltage- and temperature-based sensors attacking the shared FPGA. Giechaskiel et al. [14, 15] and Provelengios et al. [40] demonstrated that covert- and side-channel attacks could be launched on shared FPGAs fabric by utilizing ROs to observe the adjacent long wires on both Xilinx and Intel SRAM FPGAs. Zhao et al. [68] performed a SPA (simple power analysis) attack on an RSA crypto module with an RO-based power sensor. Similarly, Schellenberg et al. [45] reconstructed an AES encryption module by TDCs instead of ROs on a Spartan-6 FPGA. Tian et al. [50] showed that it is feasible to create thermal covert channels in the real cloud FPGAs by using ROs. Shayan et al. [31] demonstrated that TDCs (Time-to-Digital Converters) could be utilized to recover the input images on share FPGA instances. To the best of our knowledge, we must first carry out the model stealing attack when the adversary only has remote access to an FPGA instance. Though most of the prior works focused on attacking a single FPGA, a few recent works demonstrated it is feasible to launch "cross-FPGA" attacks. Schellenberg et al. firstly showed that by leveraging an FPGA chip as the stepping stone, the adversary could recover the core key of another chip implementing RSA and AES cryptographic modules [46]. More recently, Giechaskiel et al. [41] showed the power supply unit (PSU) can be exploited to construct covert channels across boards.

Chapter 3

Attack Methodology

In this section, we explain how DNN structure can be extracted based on RO-based power side-channel. Our attack is motivated by the observation that varying a DNN structure, including its layers and hyper-parameters, leads to changes in its power consumption, which can be exploited to infer its secret. Below, we first describe the FPGA power sensor. Then, we elaborate on the relation between DNN structure and power consumption. Finally, we overview the flow of our attack.

3.1 FPGA Power Sensor

3.1.1 Power Distribution Networks (PDN)

The power consumed by an FPGA circuit can be modeled by the sum of charging and short-circuit power consumption. To guarantee each component of the FPGA board operates rightly, an unswerving and steady voltage supply is required on the board, achieved by PDN (Power Distribution Networks) [57]. Still, PDN cannot offer a constant voltage supply for

all components. Unusual switching activities or large power consumption of a sudden would lead to power fluctuation throughout the board. To guarantee each component of the FPGA board operates rightly, an unswerving and steady voltage supply is required on the board, achieved by PDN (Power Distribution Networks) [57]. A PDN uses a voltage regulator to adjust the current and decoupling capacitors to handle current variations. Still, PDN cannot completely hide current variations. Unusual switching activities or large power consumption suddenly would lead to transient voltage drops [35]. In particular, the RO circuit is capable of recognizing the slight difference between the different voltage fluctuations. Thus, we implement RO-based power sensors that can reveal the side-channel information of other modules.

3.1.2 Ring Oscillator (RO) Power Sensor

Since the voltage fluctuation reflects FPGA logic's computation patterns, an adversary can deploy a power sensor in her slot and infer the other victim's secret. Zhao et al. [68] has demonstrated RO sensor, a circuit consisting of the odd number of inverters in a ring configuration, can be leveraged for power side-channel attacks. For example, the RO circuit is capable of recognizing the slight difference between the different voltage fluctuations. Thus, we implement RO-based power sensors that can reveal the side-channel information of other modules. Certain circuits, for example, ring oscillator circuits, are sensitive to power fluctuation, and their frequency could be affected, which can be exploited for power side-channel attack. As mentioned, by can be leveraged as the power sensor. In fact, the voltage fluctuations have a prominent impact on the frequency of RO, which can be read through T *flip-flops* circuit. In Figure 3.1, we illustrate the design of our RO sensor. For each power sensor, three inverters are connected sequentially along with an AND gate. The output of the last inverter is combinational fed back to the input of the AND gate, which constitutes a ring configuration. An enable signal is connected to the AND gate as another input. To

measure the frequency of the RO, we connect the output of the last inverter to a 16-bit T flip-flops counter. The reading from the counter is sent to a workstation for further analysis through *xillybus* [59], which is an FPGA IP core for data transmitted over PCIe[1]. To improve the robustness of readings and get a satisfying resolution with a given sampling period of 100 Mhz on Zedboard, we implement 20 RO sensors and connect their output to an adder tree to obtain their sum as the final power sensor value. Also, for other alternative designs of RO, we discuss their performance in Section 4.5. In Section 4, we discuss the impact caused by the number of RO sensors.

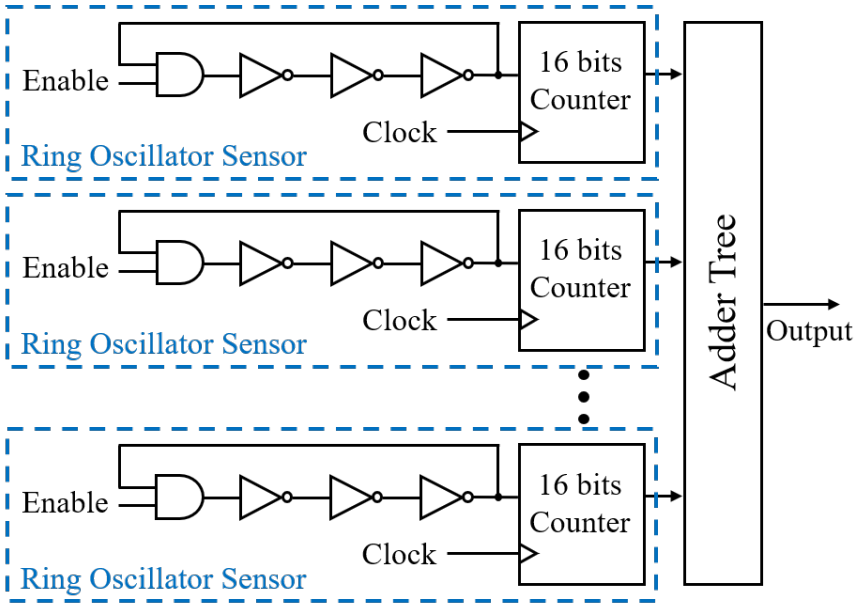

Figure 3.1: Ring Oscillator Power Sensors

3.1.3 Placement of RO Power Sensor

How RO sensors are placed on FPGA impacts the attacker's readings, further influencing the attack accuracy. Because the FPGA logic of different users can share the same FPGA board, cloud providers enforce placement policies for every user. Figure 3.2 and Figure 3.3 illustrate the floorplan related to different policies when one user is an attacker: free placement and

[1]On FPGA cloud instances like Amazon F1, PCIe Physical Functions (PFs) are provided to tenants for accessing and controlling their FPGA logic [2].

restricted placement. In Section 4.5, we evaluate how the placement policy impacts attack accuracy.

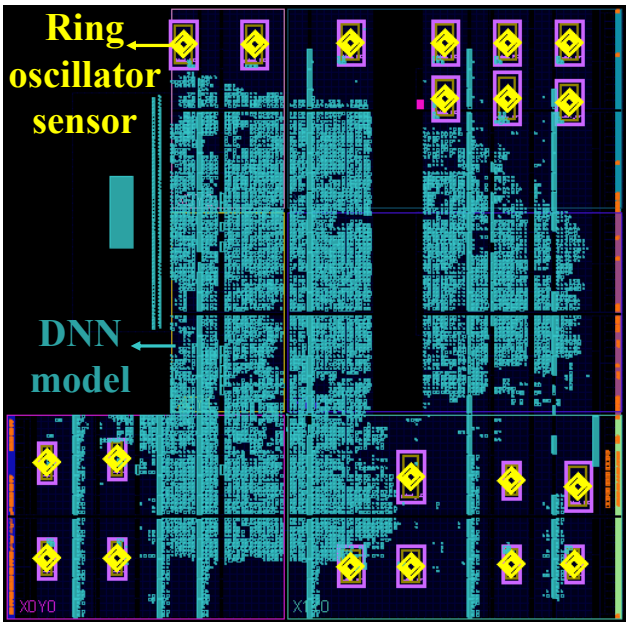

Figure 3.2: The floorplan of FPGA when the attacker has permission to determine the placement of RO sensor

For the first one, the attacker has permission to deploy FPGA logic wherever she desires on the board. The attacker would prefer to place the malicious logic *close* widely and evenly distributed on the whole board, providing the attacker with better resolution of power sensor readings. For attack simulation, we distributed 20 ROs evenly throughout the FPGA board (see Figure 3.2, utilizing the Pblocks constraints (one approach for floorplan FPGA design) provided by Xilinx Vivado.

For the second set, the attacker has no control over the location of RO power sensors. In this case, the FPGA instance is separated into several virtual FPGAs, and the cloud provider decides where each logic is placed. We simulate this case by constraining all ROs in one Pblock, so they are placed in one virtual FPGA slot, similar to the constrained placement requirement in the *FPGA multi-tenancy* scenario.

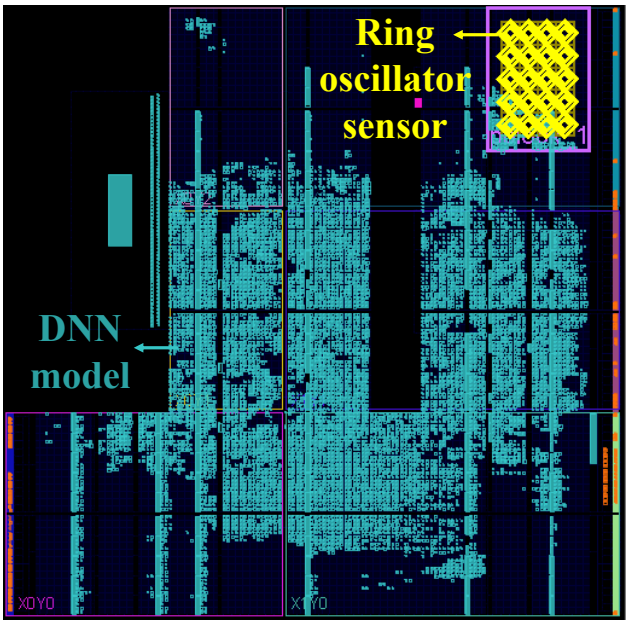

Figure 3.3: The floorplan of FPGA when the attacker has **no** permission to determine the placement of RO sensor

3.2 DNN Layers and Computational Workload

Deep neural networks are a family of methods that transfer input to output with various repetitious processing units. Each processing unit is called a *layer*. The *layer*, which is always referred to as the repetitious processing units and whole transformation from input to output, is called a DNN model. The full set of layers, including their hyper-parameters settings, are named DNN *structure*. For a power sensor to discern structural elements of a DNN model, each element should incur a different computation workload. Below we analyze the three most popular layers' workload: fully-connected, convolutional, and pooling layers. Each layer is also associated with a set of hyper-parameters, and we list them in Table 3.1. The equations about the workload analysis are listed in Equation 3.1 to 3.4 at the end of this subsection. Equation 3.5 lists an important constraint about hyper-parameters, which is leveraged for result correction, as described in Section 3.3.

Table 3.1: Hyper-parameters of neural network

Layer Type	**Hyper-Parameter**	**Definition**
FC layer	N_i	Number of neurons of $layer_i$
Conv layer	W_i	Width of output feature map of $layer_i$
	F	Size of filter
	D_i	Depth of output feature map (Number of filters)
	S	Stride
Pooling layer	W_i	Width of output feature map of $layer_i$
	F	Size of filter
	S	Stride

3.2.1 Fully-connected (FC) Layer

A fully connected layer consists of many neurons. Each one computes a weighted sum on all neurons from the previous layer, followed by a bias offset (β) and activation function (ϕ) like ReLu. The output can be represented as $\phi(N_{i-1} \times W_i + \beta)$, where N_{i-1} is the vector of neurons of the prior layer, W_i is the weight matrix and $\times$ is the vector-matrix multiplication. FPGA can accelerate the computation through optimized *MAC (multiply and accumulation)* operation. The number of MAC operations FC_{mac} done by each layer can be derived using Equation 3.1.

$$FC_{mac} = N_{i-1} \times N_i \tag{3.1}$$

3.2.2 Convolutional (Conv) Layer

A convolutional layer carries out *convolution* operation between the input layer and its kernels, followed by bias offset and activation function. Unlike the fully connected layer, each neuron in the convolutional layer generates an output value using a *region* of neurons from the prior layer. The weight matrix applied to the region is called *kernel*. The input can have multiple channels (e.g., an image can have red, green, and blue channels), and a *filter* aggregates kernels' output across channels. During convolution, the filter slides over

the input data, and each step is measured by *stride*. When the filter goes over the edge of the input, *padding* is performed to extend the input region. Like the fully connected layer, MAC is also leveraged to accelerate the convolutional layer, and the computational workload of the convolutional layer depends on how many MAC has been executed. Equation 3.2 shows the number of MAC operations, which is determined by W_i, F, D, S and P. They refer to the size of the output feature map, size of the filter, the depth of output volume (number of filters), stride, and padding [2].

$$CONV_{mac} = {W_i}^2 \times F^2 \times D_{i-1} \times D_i \tag{3.2}$$

Except for W_i, all other inputs are hyper-parameters set by the model developer. W_i can be computed through Equation 3.3.

$$W_i = \frac{W_{i-1} - F + 2 \times P}{S} + 1 \tag{3.3}$$

3.2.3 Pooling Layer

The pooling layer is a special type of convolutional layer. Its main function is to reduce the spatial size of the prior layer's output, also controlling overfitting. A pooling layer includes multiple filters, and each filter aggregates an input region to a scalar value, usually through MAX operation. The workload of the pooling layer can also be computed using Equation 3.2, except that MAC is replaced by MAX and W_i is replaced by $W_{pooling}$ (see Equation 3.4).

$$W_{pooling} = \frac{W_{i-1} - F}{S} + 1 \tag{3.4}$$

[2]We ignore padding in Equation 3.2 for simplicity. Padding typically ranges from 0 to 3, which negatively impacts the overall MAC number.

3.3 Attack Flow

From the prior analysis, it is clear that different layers introduce different types of operators (e.g., MAC by fully-connected layer and MAX by pooling) and workload. Therefore, we use power traces collected by the spy to infer the workload of the victim DNN sharing the FPGA board and further reveal its structural secret.

Before the model-stealing attack, the attacker needs to train an inference model (M) by profiling a set of DNN models with different layers and hyper-parameter compositions. During the attack, the power traces are segmented by M, and the layer and hyper-parameters of each segment are predicted. To notice, the targeted model and the profiled models do not have to belong to the same DNN family (e.g., VGG). The major requirement is that the layers and hyper-parameters have been "seen" in the profiled models. The attack flow is shown in Figure 3.4, and we elaborate on each stage below.

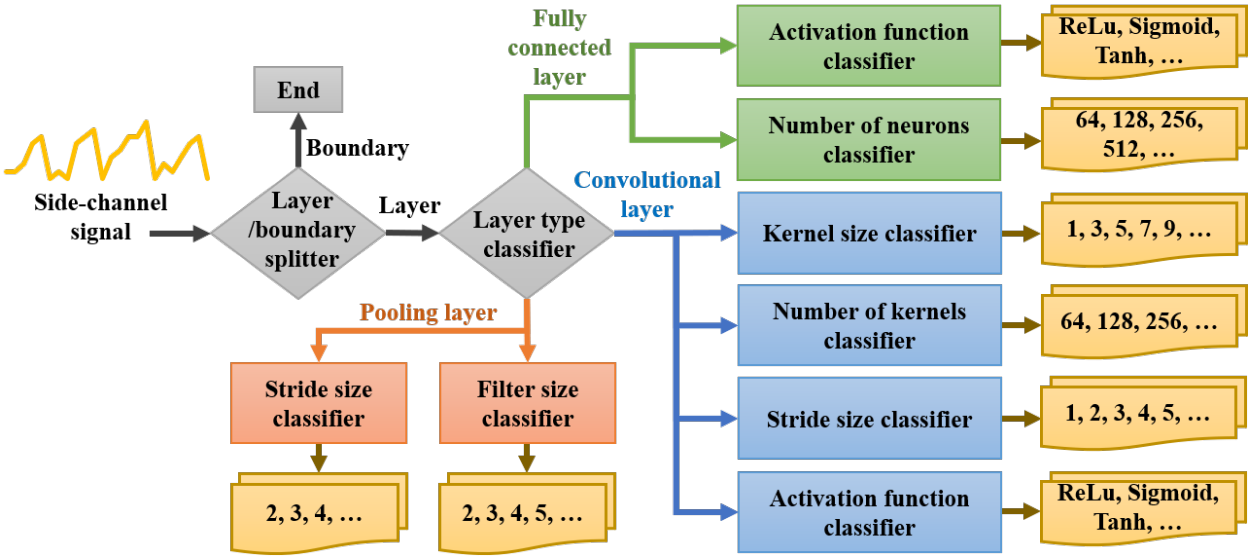

Figure 3.4: Attack Workflow

3.3.1 Layer Segmentation

Before FPGA computes a layer, it would wait for the CPU to pass the layer configuration and weights to the on-board DRAM. We name such a short waiting period as *"boundary"*, which usually causes less power consumption. We differentiate the layer and boundary with a binary classification model. Figure 3.5 illustrates the readings of *"boundary"* and *"layer"*. After this step, the number of layers would also be revealed.

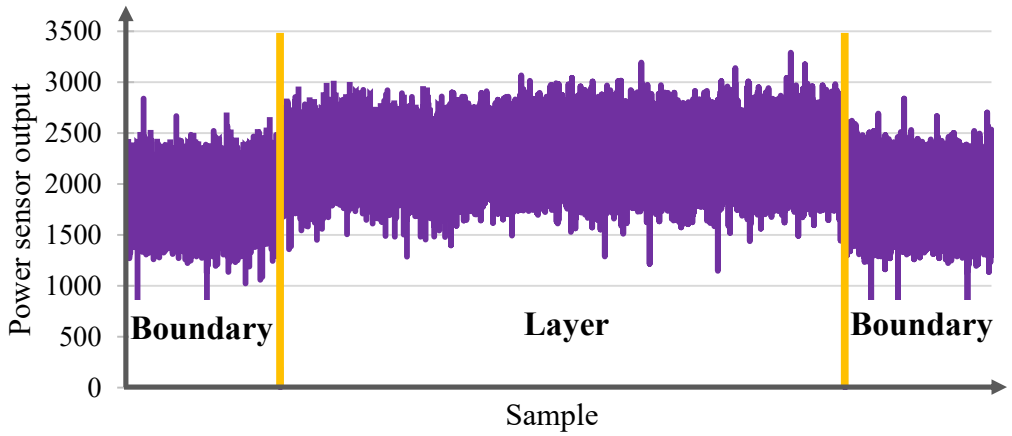

Figure 3.5: The output signal of RO sensors under convolutional layer

3.3.2 Recognizing Layer Type

After the attacker finishes the first step, the sequence of power traces is separated by layers. For each segment of power traces, the attacker classifies it to a fully-connected, convolutional, or pooling layer. We consider each segment as a time-series and extract a number of statistical features before classification. For a fully-connected layer, it conducts a matrix multiplication, while for the convolutional layer, it completes the convolution between multiple filters and input feature maps. Based on the equation of 3.1 and 3.2, these two types of the layer would generate different power trace. Finally, for the pooling layer, this layer applies a MAX function on the previous feature map to get the maximum value of filter on the feature map,

which is a different function compared to the fully-connected and convolutional layer.

3.3.3 Recognizing Hyper-parameters

Different layer type embodies different sets of hyper-parameters. For a fully-connected layer, the hyper-parameter is the number of neurons. For a convolutional layer, hyper-parameters include filter size, the number of filters, and stride. As shown in Equation 3.1, 3.2, and 3.4, the hyper-parameter values have a direct impact on the computing workload of a layer so that we can infer them from the power traces. Though theoretically, victims can choose any value for the hyper-parameters, nowadays, for the best DNN performance, those values usually fall in a small range (e.g., padding usually ranges from 0 to 3, and the number of neurons typically is a multiple of 2). Therefore, we can profile their different combinations ahead and detect their existence later with a classification model. In addition to the hyper-parameters with numerical values, each layer's activation function is also detected at this stage.

3.3.4 Correction with Model Constraints

After the above steps, the attacker obtains the model structural secret. Although our classification models can make the right prediction most of the time, misclassifications still happen. On the other hand, some of them can be corrected using constraints *inherent* in DNN model design. The hyper-parameters of one layer can be corrected, based on the adjacent layers, as shown in Equation 3.3 and Equation 3.5.

$$S \leq F \leq \frac{W_i}{2} \tag{3.5}$$

As one example shown in Equation 3.5, F should be lower-bounded by S and upper-bounded by $\frac{W_i}{2}$. Therefore, when the predicted F is over S, it can be scaled back to S. In addition to leveraging the 5 Equations shown in Section 3.2 for the result correction, we can also leverage the advice widely accepted in the machine-learning community. For instance, the number of filters usually increases as the layer depth grows (often doubles the size). Therefore, the filter number that falls out of the range can be corrected.

Chapter 4

Evaluation

In this section, we first describe the setup for our evaluation. Then, we elaborate on each attack step's result and the end-to-end accuracy of the targeted models. Additionally, we discuss how the number and placement of RO impact the result, how much benefit the attacker could gain from the inferred model, and the other types of power sensors. Finally, we assessed how effective the recovered model is on a testing dataset.

4.1 Experiment Setup

We run experiments on a ZedBoard [58], which contains a dual Corex-A9 Processing System (PS) with 85,000 Series-7 Programmable Logic (PL) cells. Similar to Hua et al. [20], the layers are executed sequentially. For most of the tasks, we deploy 20 RO power sensors and follow the floorplan shown in Figure 3.2. For the RO power sensor's implementation, we utilize the typical FPGA design flow provided by Xilinx Vivado. All components of RO sensors are written in Verilog, a hardware description language. During the synthesis procedure, the developer can give each design component the placement constraints by a

.xdc file or Pblock. In our case, we decide to use Pblock to confine the location of each RO sensor. After synthesis, Xilinx Vivado translates the register-transfer level (RTL) code into the bitstream file, containing the circuit implementation information for the FPGA board. We assume the victim DNN runs in the *testing* mode. Inferring secret when DNN is *training* is also feasible under our attack, as training only adds a back-propagation process after a batch of input data goes through all layers.

To train our inference model M, we constructed 30 variants of AlexNet with customized layer and hyper-parameter settings and collected their power traces. We profile variants of AlexNet because they contain nearly all elements (except non-sequential connection) used by other recent DNN models. AlexNet is a symbolical family of DNN structure, which achieved first place in the ImageNet [11] 2012 Challenge. After AlexNet, numerous CNN architectures were built based on the foundation of AlexNet. For example, the ZFNet [65], which is the champion of ImageNet 2013 Challenge, has the same layer types as AlexNet but improved it by adopting new hyper-parameters. Also, AlexNet embraces both fully-connected layers and convolutional layers, which are straightforward to be customized for all hyperparameter settings. For the convolutional layer, the filter size is set to be an odd number ranging from 1 to 13, the size of stride ranges from 1 to 5, and the number of filters ranges from 64 to 8192. For the fully-connected layer, the number of neurons is a multiple of 2, varying from 64 to 4096. For the pooling layer, the filter size is set to range from 2 to 5, and the size of stride varies from 2 to 4. For the workload to be executed by the DNN models, we sample one image from ImageNet [11] (resized to 224x224) and test it repeatedly. Our attack is input-agnostic as the same amount of operations has to be executed during inference for any input. We run AlexNet on the FPGA for 50 iterations for each hyper-parameter value, collecting over 1 million readings from RO sensors.

To evaluate the effectiveness of M, we test it against models under the same family (AlexNet) and **also models from other families**, including VGG16 and 5-layer MLP (Multilayer

perceptron). Each tested model is executed 20 times with the same input, and we aggregate the accuracy across models. For the final end-to-end analysis, we report the accuracy for each tested model individually.

4.2 Layer Segmentation

We use XGBoost [7], a tree boosting model, to classify power traces into *"layer"* and *"boundary"*. We applied the classification model on each data point initially, but the result is unsatisfactory, with only 61.83% accuracy on the testing data. Therefore, we introduce a *voting* procedure to improve the accuracy: we group the data points by sliding window (stride is set to 1) and use the majority decision as to the classification result.

We have tested different window sizes, ranging from 50 to 400, with our power traces collected during training (AlexNet). As shown in Figure 4.1, when window size equals 300, it reaches the peak accuracy, at **96.57%**. As a result, we choose 300 as the window size. When there are more than 3 consecutive windows predicted as the boundary, we consider the data points covered by them related to the layer boundary and the remaining ones related to the layers.

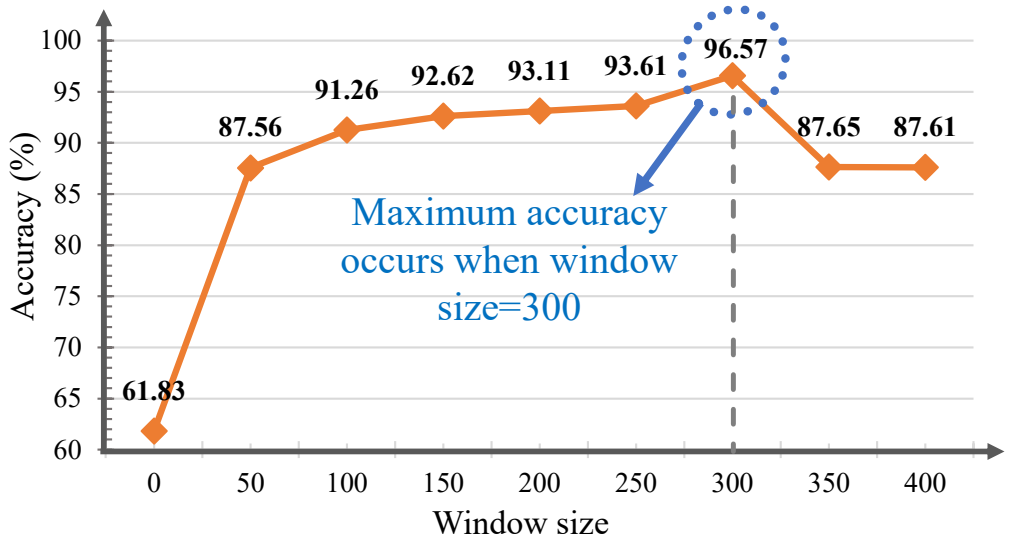

Figure 4.1: The accuracy of layer segmentation for different window sizes

4.2.1 Feature Selection

After finishing the layer segmentation, we treat all data points between two *"boundaries"* related to *"layer"* and classify its type. As the data can be treated as time series, we extract their features leveraging `tsfresh` [8], a python library that derives hundreds of features automatically from a time series. The `tsfresh` library provides features of time-domain, frequency-domain, and wavelet-based, but not all of them are useful: for example, some of them are always zero and constant. Thus, we rank the features and select the ones under the Mann–Whitney U test. If the p-value is less than 0.001, this feature is statistically significant and selected. We choose 28 features and Table 4.1 lists the top 10 features.

Table 4.1: Selected features (Top 10)

Features	**P_value**	**Description**
Linear trend	2.29e-07	Calculates linear regression for data versus sequence.
Agg linear trend(mean)	7.43e-07	Calculates linear regression for aggregated data versus sequence(aggregate function is mean).
Agg linear trend(min)	1.34e-06	Calculates linear regression for aggregated data versus sequence(aggregate function is min).
ABS energy	1.42e-05	Calculates absolute energy of data
Sum values	1.45e-05	Calculates sum over time series values
Mean	1.45e-05	Returns the mean of data
FFT coefficient	1.45e-05	Calculates the fourier coefficients of data
Sum of reoccurring data points	1.53e-05	Returns the sum of data points that present more than once
Quantile q=0.9	4.94e-05	Calculates the q quantile of data(q = 0.9)
Quantile q=0.6	1.10e-05	Calculates the q quantile of data(q = 0.6)

4.2.2 Layer Identification

After layer segmentation, the next step is to recognize the type of each layer, namely convolutional layers, fully-connected layers, and pooling layers. We examined **8** classification

models for this task on the selected 28 features, and the result of the testing data shown in Table 4.2 (Column "Layer type").

Table 4.2: Accuracy of the classification models using selected features

Classifiers	**Settings**	**Accuracy(%)**					
		Layer type	Number of neurons	Filter size	Number of filters	Stride	Activation function
Nearest Neighbors [32]	$n\_neighbors = 5$	87.5	35	37.14	37.14	86.67	100
Gradient Boosting Classifier [13]	$n\_estimators = 1000$	97.5	**96.67**	91.42	71.42	96.67	96.67
Decision Tree[44]	-	97.5	94.67	68.57	68.57	96.67	100
Random Forest [25]	$n\_estimators = 1000$	**100**	71.67	57.14	31.42	93.33	100
Multi-layer Perceptron classifier[38]	$activation\_function = relu, hidden\_layer\_sizes = 1000, solver = adam$ [22]	90	15	14.28	20	73.33	90
Naive Bayes [42]	-	87.5	23.33	17.14	20	90	90
AdaBoost [17]	-	97.5	40	31.42	31.42	**97.47**	100
XGBClassifier [7]	$n\_estimators = 1000, max\_depth = 5$	**100**	93.33	**94.28**	**94.85**	96.67	**100**

To learn the impact of feature selection, we run a comparison experiment: each layer segment is padded to the same size as the longest layer. All data points are fed into classifiers. The accuracy of every classifier is much lower (e.g., less than 40% accuracy) than the accuracy with feature selection. Thus, the feature extraction step is needed before classification. After that, we selected Nearest Neighbors, Gradient Boosting, Decision Tree, Random Forest, Multi-layer Perceptron classifier, Naive Bayes, AdaBoost, XGBoost as classifier candidates. The settings for each classifier is also listed in Table 4.2. Among all classifiers, tree-based models like XGBoost and RandomForest perform best, reaching **100%** accuracy in this task.

4.3 Hyper-parameter Identification

After the spy figures out the layer type, it can extract all hyper-parameters associated with each layer. We tested the same set of 8 classification models in this task, and the result is shown in Table 4.2. The number of neurons is the secret for the fully-connected layer, and we found Gradient Boosting reaches the optimal performance, with 96.67% accuracy. For convolutional layer and pooling layer, filter size, the number of filters, stride and activation function (including ReLu, Sigmoid and Tanh) are secret, and we found high accuracy (**94.28%**, **94.85%**, **97.47%** and **100%**) can be achieved. Overall, a quite high accuracy (over 93%) can be achieved to infer any secret with the tree-based classifiers. We use the XGBoost classifier for the end-to-end inference attack on a targeted model as it achieves the best performance for most inference steps (4 out of 6).

4.4 End-to-end Result

Finally, we assemble the inferred layers and their hyper-parameters to form a predicted *layer sequence* and compare it to the ground truth. The accuracy here is defined as the number of correctly predicted elements (layer and hyper-parameter together) overall elements. The result is summarized in Table 4.3.

As each model is tested for 20 traces, we show the optimal one for the predicted sequence, and the accuracy is averaged across all 20 traces. For MLP model, we use 5 different neuron numbers (64, 128, 128, 256, 512) for the 5 layers and we can reach **100%** accuracy. For the AlexNet model, four mistakes are observed on hyper-parameters, but the layers are all correctly predicted, resulting in 91.43% accuracy overall. For the VGG16 model, we can reach 91.78% accuracy (7 mistakes) for the sequence prediction. The number of filters is mistakenly classified initially, but they can be corrected using the model constraints described

Table 4.3: Result for end-to-end layer sequence prediction (C=`Conv` and the subscripts of C stand for the size of the filter, number of filters, and stride. F=`fully-connected` and the number of neurons in F are shown in subscript. P=`Pooling`, and its subscripts represent filter size and stride. The activation function is omitted due to space limitations. Accuracy is computed after model correction. Red characters represent misclassifications while the ground truth in green.)

Model	Original layer Sequence Predicted layer Sequence	*Accuracy*
MLP	$F_{64} - F_{128} - F_{128} - F_{256} - F_{512}$ $F_{64} - F_{128} - F_{128} - F_{256} - F_{512}$	100%
AlexNet	$C_{11,96,4} - P_{3,2} - C_{5,256,1} - P_{3,2} - C_{3,384,1} - C_{3,384,1} - C_{3,384,1} - P_{3,2} - F_{256} - F_{256} - F_{64}$ $C_{11,128,4} - P_{3,2} - C_{3,256,1} - P_{3,2} - C_{3,384,2} - C_{3,384,1} - C_{3,384,1} - P_{3,2} - F_{256} - F_{256} - F_{64}$	91.43%
VGG16	$C_{3,64,1} - C_{3,64,1} - P_{2,2} - C_{3,128,1} - C_{3,128,1} - P_{2,2} - C_{3,256,1} - C_{3,256,1} - C_{3,256,1} - P_{2,2} - C_{3,512,1} - C_{3,512,1} - C_{3,512,1} - P_{2,2} - C_{3,512,1} - C_{3,512,1} - C_{3,512,1} - P_{2,2} - F_{512} - F_{256} - F_{128}$ $C_{3,64,1} - C_{3,64,1} - P_{2,2} - C_{3,128,1} - C_{3,128,1} - P_{2,2} - C_{3,512,1} - C_{3,512,1} - C_{3,256,1} - P_{2,2} - C_{3,512,2} - C_{3,512,2} - C_{3,512,1} - P_{2,2} - C_{3,512,1} - C_{3,512,1} - C_{3,512,1} - P_{2,2} - F_{512} - F_{256} - F_{128}$	94.52%

in Section 3. Therefore the final result is improved to 94.52%.

4.5 Impact of RO Power Sensor Configurations

In this section, we firstly explored how many RO power sensors should be injected on the board to achieve the best outcome of the DNN model recovery. Then we consider different placement policies that will impact the accuracy of our attack. Finally, we discuss the performance of other alternative designs of RO power sensors.

4.5.1 Number of RO Power Sensors

We change the number of RO power sensors from 1 to 20 to assess its impact on the attack accuracy. For simplicity, we only show the result of layer segmentation in Figure 4.2, as the trends for other inference tasks are similar. When the number of ROs rises from 1 to 15, the accuracy grows from 73.17% to 95.34%. After 15, the improvement of accuracy is small. Therefore, the adversary can reduce the RO amount to 15 if the FPGA resource is constrained without sacrificing much of the inference accuracy. Finally, we choose to implement 20 RO sensors on the board.

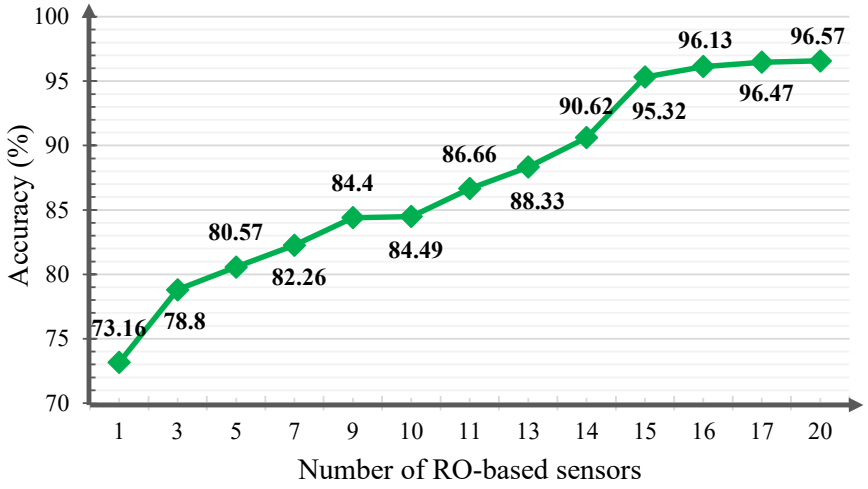

Figure 4.2: The accuracy of layer segmentation for different number of sensors

4.5.2 Placement of RO Power Sensors

We consider the two placement policies for RO power sensors. As Figure 3.2 shown, in this situation, the attacker can constrain the placement of each RO power sensor wherever she wants. The second scenario, like Figure 3.3 represents that each tenant on this board would be restricted to her own virtual FPGA slot of the board. So we implement 20 RO into one Pblock to ensure they are restrained into the same slot. We experiment with the second scenario by collecting one million data of boundary and layer and estimate the accuracy to

classify between them. As a cloud provider might prohibit its user from freely placing FPGA logic, we examine how this strategy impacts the accuracy. We place and route 20 ROs as Figure 3.3 shown and run the same attack for layer segmentation. Surprisingly, we found that our attack can still differentiate between boundary and layer successfully only with comparatively minor accuracy decreases(drop from 96.57% to 86.63%). We believe that this is because our attack relies on 20 RO power sensors readings, making variations in individual RO negligible.

4.5.3 Alternative RO Power Sensors

The power sensor we use for the attack is based on the conventional RO sensor, which the prior work has also used for remote FPGA attack [68]. On the other hand, we found cloud providers are examining the combinatorial loops in the logic before it is instantiated on the FPGA board [3]. The adversary would need to change the design of the sensor to avoid detection. Specifically, we found two alternative designs, Latch-based RO sensor [15] and Flip-Flop-based RO sensor [48], that is capable of achieving a similar sampling resolution as the conventional RO sensor. The architectures of these two ROs are shown in Figure 4.3 and Figure 4.4, which replaces an inverting buffer with a latch and a flip-flop, respectively.

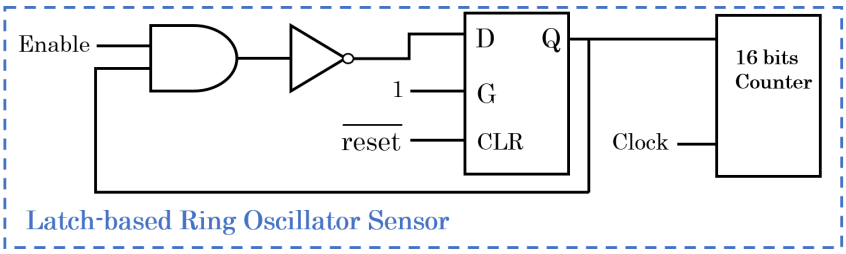

Figure 4.3: Latch-based Ring Oscillator Sensor

We evaluate how the attack accuracy changes when they are used. We focus on the task of layer segmentation, which is the first step of the attack flow in Section 3.3. We retrained

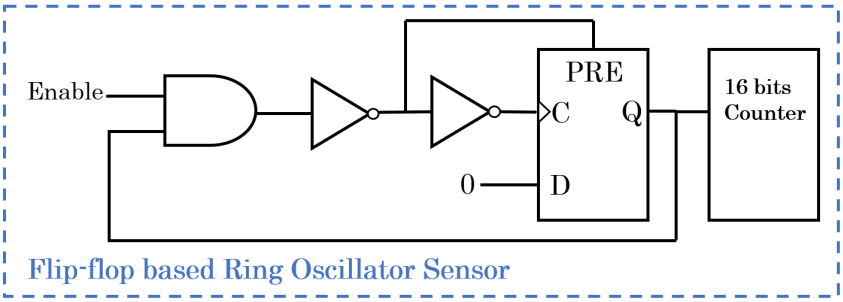

Figure 4.4: Flip-flop based Ring Oscillator Sensor

layer/boundary classifiers using the RO sensor readings from Latch-based RO and Flip-Flop based RO separately and evaluated on the testing readings. The readings are also grouped into windows of 300 data points. In the end, the accuracy on the layer segmentation task can reach 97.14% by Latch-based RO and 95.84% by Flip-Flop based RO, which are comparable to the conventional RO (96.57%), suggesting the adversary has flexibility in switching between RO designs to make the attack stealthy. In addition to two alternative RO power sensors, the delay-line monitor [45] can also be exploited as a power sensor. But as studied by Zhao et al. [68], compared to the RO sensor, it is less stable and has lower power resolution. Therefore, we choose RO sensors as our power trojan for most of the evaluation tasks.

4.6 Effectiveness of Recovered Models

While we successfully recovered the layer structure and hyper-parameters for each layer at high accuracy, how much benefit the adversary can gain is yet unclear. We try to answer this question by comparing the classification accuracy using the original model and the recovered model (with the structure only). Specifically, we programmed the original and recovered version of AlexNet (illustrated in Figure 2.2) using the layer sequences shown in Table 4.3) with pytorch-V1.5 [37] and NVidia CUDA-V10.2 [34]. The models are trained and

tested on Tiny ImageNet-200 dataset [9], which comprises 200 image classes. Each image class has 500 training images, 50 validation images, and 50 test images. Using this dataset allows us to assess the classification accuracy at a much smaller overhead than the original ImageNet dataset. Some changes are also applied to the model input. For example, AlexNet is developed for the classification of RGB images with 224 x 224 pixels. However, the images in the Tiny ImageNet-200 dataset are 64 x 64 pixels. Thus, in the data prepossessing phase, the dataset images were scaled to 224 x 224 pixels with normalization. We set 256 and 0.01 as batch size and the learning rate and use Adam optimizer [22] for training. The number of epochs for training is set to 150. In the testing stage, we compare the top-1 accuracy and top-5 accuracy (i.e., how likely the top one and top five prediction matches the label of the tested image). The comparison of accuracy is shown in Table 4.4, suggesting the adversary's workload in model tuning will be significantly reduced when starting from the stolen model.

Table 4.4: Accuracy of original and recovered model

Model		**Top-1 Accuracy**	**Top-5 Accuracy**
AlexNet	Original	31%	57%
	Recovered	24%	55%

By exploiting the inferred model structure, the attacker can also launch an adversarial attack [36] such as evasion more effectively. Hu et al. [19] showed that the success rate could be increased by more than 50% even when a few inferred layers are wrong, compared to the black-box adversary who has zero knowledge about targeted models. We expect a similar or better success rate can be gained under our attack method.

Chapter 5

Attack Mitigation

We discuss the limitations of our attack method and the potential countermeasures in this section.

5.1 Limitations

While our attack is demonstrated effectively for *FPGA multi-tenancy* scenario, main cloud FPGA providers only support *FPGA single-tenancy* for now. To make our attack feasible on the commercial cloud now, one option is to conduct *cross-FPGA* power measurement. In fact, very recently, Giechaskiel et al. [41] proved the power supply unit (PSU) can be exploited to construct FPGA-to-FPGA, CPU-to-FPGA, and GPU-to-FPGA covert channels between different boards. We plan to simulate their settings and assess whether cross-FPGA model-stealing attack is practical.

Our attack focuses on DNN structure, leaving neuron weights not inferred. Hua et al. [20] showed weights can be inferred, but they assume the adversary can feed input to the CNN inference accelerator. They also assume "zero pruning" is done on the feature map of the

CNN model, which is not always the case. In future work, we will keep exploring weights inference, e.g., by evaluating the attack when part of the input data is public and known to the adversary.

We evaluated our attack on one FPGA board, ZedBoard, due to budget and time limitations. We did not test with other FPGAs. Still, we believe that the attack should also work on other FPGAs, as our attack setup is not specific to any FPGA board.

Our attack can recover the majority of common hyper-parameters. When uncommon hyper-parameters are used (e.g., the number of neurons are not the multiples of 2), our attack result could be inaccurate. To address this issue, the adversary can enlarge the set of hyper-parameters to be profiled. On the other hand, those uncommon settings usually degrade the model performance, and we believe they are doubtful to be adopted.

5.2 Countermeasures

Our attack could be thwarted by two countermeasures potentially: (1) hiding the power consumption patterns unique to the DNN layers/hyper-parameters or (2) rejecting the deployment request of suspicious FPGA logic. The first schema requires supplement hardware[39] to be installed to mask the power consumption (e.g., through noise injection) of every operation or embed active fences[23] surrounding the logic of all users on the cloud FPGA. However, this approach will incur extra execution overhead unavoidably, which may eventually offset the benefit brought by the DNN acceleration from FPGA. The second method requires the cloud provider to verify each tenant's RTL (Register Transfer Level) design. However, this countermeasure requires the provider has a "blacklist" about the logic to be rejected [3]. In Section 4.5.3, we showed that by switching from the conventional RO design to Latch-based RO or Flip-Flop-based RO designs, similar attack accuracy could be

achieved. So far, we have not found a bullet-proof detection approach against the power trojans. In other words, a solution that can provably mitigate our attack without incurring high overhead seems non-trivial.

Chapter 6

Conclusion

This study investigates whether the structural secret (layers and hyper-parameters) of a victim DNN model can be inferred by a remote attacker who shares the same FPGA board. We show by implementing on-chip RO-based power monitors, the power consumption of the victim DNN can be sampled at high resolution, which guarantees all layers and hyper-parameters can be reconstructed at high accuracy. The result of our attack suggests the intellectual property of a machine-learning company, or DNN model secret, should be carefully guarded when using the cloud computing resources, even when physical and logical isolation is enforced. We call the security community's attention to develop new cost-effective defense solutions against this threat, and we plan to investigate it as well.

Bibliography

[1] A. A. Al-Aghbari and M. E. Elrabaa. Cloud-based fpga custom computing machines for streaming applications. *Ieee Access*, 7:38009–38019, 2019.

[2] Amazon. Amazon ec2 f1 instances. `https://aws.amazon.com/ec2/instance-types/f1/`.

[3] Amazon. Aws ec2 fpga hdk+sdk errata. `https://github.com/aws/aws-fpga/blob/master/ERRATA.md`.

[4] L. Batina, S. Bhasin, D. Jap, and S. Picek. Csi nn: reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Conference on Security Symposium*, pages 515–532, 2019.

[5] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 109–116. IEEE, 2014.

[6] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang. Enabling fpgas in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, pages 1–10, 2014.

[7] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, 2016.

[8] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[9] S. CS231N. Tiny imagenet visual recognition challenge. `https://tiny-imagenet.herokuapp.com/`, 2015.

[10] G. Dai, Y. Shan, F. Chen, Y. Wang, K. Wang, and H. Yang. Online scheduling for fpga computation in the cloud. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 330–333. IEEE, 2014.

[11] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[12] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas. Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720*, 2018.

[13] J. H. Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[14] I. Giechaskiel, K. Eguro, and K. B. Rasmussen. Leakier wires: Exploiting fpga long wires for covert-and side-channel attacks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(3):1–29, 2019.

[15] I. Giechaskiel, K. Rasmussen, and J. Szefer. Reading between the dies: Cross-slr covert channels on multi-tenant cloud fpgas. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 1–10. IEEE, 2019.

[16] Google. Google compute engine(gce). `https://cloud.google.com/compute/`.

[17] T. Hastie, S. Rosset, J. Zhu, and H. Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[18] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitraş. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. *arXiv preprint arXiv:1810.03487*, 2018.

[19] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. 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.

[20] W. Hua, Z. Zhang, and G. E. Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[21] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori. Active fences against voltage-based side channels in multi-tenant fpgas. *IACR Cryptol. ePrint Arch.*, 2019:1152, 2019.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[25] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[26] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4–22, 1987.

[27] S. Mangard. A simple power-analysis (spa) attack on implementations of the aes key expansion. In *International Conference on Information Security and Cryptology*, pages 343–358. Springer, 2002.

[28] J. M. Mbongue, F. Hategekimana, D. T. Kwadjo, and C. Bobda. Fpga virtualization in cloud-based infrastructures over virtio. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 242–245. IEEE, 2018.

[29] J. M. Mbongue, A. Shuping, P. Bhowmik, and C. Bobda. Architecture support for fpga multi-tenancy in the cloud. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 125–132. IEEE, 2020.

[30] Microsoft. Azure fpga inference. `https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-deploy-fpga-web-service`.

[31] S. Moini, S. Tian, J. Szefer, D. Holcomb, and R. Tessier. Remote power side-channel attacks on cnn accelerators in fpgas, 2020.

[32] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[33] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2139–2153, New York, NY, USA, 2018. Association for Computing Machinery.

[34] Nvidia. Cuda zone. `https://developer.nvidia.com/cuda-zone`, 2019.

[35] S. Pant. *Design and Analysis of Power Distribution Networks in VLSI Circuits*. PhD thesis, 2008.

[36] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[39] T. Popp, S. Mangard, and E. Oswald. Power analysis attacks and countermeasures. *IEEE Design & test of Computers*, 24(6):535–543, 2007.

[40] G. Provelengios, C. Ramesh, S. B. Patil, K. Eguro, R. Tessier, and D. 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, 2019.

[41] K. Rasmussen, I. Giechaskiel, and J. Szefer. Capsule: Cross-fpga covert-channel attacks through power supply unit leakage. In *IEEE Symposium on Security and Privacy*, volume 1. IEEE, 2020.

[42] I. Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[43] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, 2009.

[44] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[45] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori. An inside job: Remote power analysis attacks on fpgas. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1111–1116. IEEE, 2018.

[46] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori. Remote inter-chip power analysis side-channel attacks at board-level. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2018.

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka. Oscillator without a combinatorial loop and its threat to fpga in data centre. *Electronics Letters*, 55(11):640–642, 2019.

[49] N. Tarafdar, T. Lin, D. Ly-Ma, D. Rozhko, A. Leon-Garcia, and P. Chow. Building the infrastructure for deploying fpgas in the cloud. In *Hardware Accelerators in Data Centers*, pages 9–33. Springer, 2019.

[50] S. Tian and J. Szefer. Temporal thermal covert channels in cloud fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 298–303, 2019.

[51] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 601–618, USA, 2016. USENIX Association.

[52] A. Vaishnav, K. D. Pham, and D. Koch. A survey on fpga virtualization. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 131–1317. IEEE, 2018.

[53] B. Wang and N. Z. Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52, 2018.

[54] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf. Enabling fpgas in hyperscale data centers. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1078–1086. IEEE, 2015.

[55] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137. IEEE, 2020.

[56] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406, 2018.

[57] Xilinx. 7 series fpgas pcb design guide. `https://www.xilinx.com/support/documentation/user_guides/ug483_7Series_PCB.pdf`.

[58] Xilinx. Zedboard. `https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html`.

[59] Xillybus. Xillybus product brief. `http://xillybus.com/downloads/xillybus_product_brief.pdf`.

[60] M. Yan, C. W. Fletcher, and J. Torrellas. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2003–2020, 2020.

[61] K. Yoshida, T. Kubota, M. Shiozaki, and T. Fujino. Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 318–318. IEEE, 2019.

[62] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin. Deepem: Deep neural networks model recovery through em side-channel information leakage.

[63] H. Yu, K. Yang, T. Zhang, Y.-Y. Tsai, T.-Y. Ho, and Y. Jin. Cloudleak: Large-scale deep learning models stealing through adversarial examples. In *Proceedings of Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[64] X. Yuan, L. Ding, L. Zhang, X. Li, and D. Wu. Es attack: Model stealing against deep neural networks without data hurdles, 2020.

[65] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[66] K. Zhang, Y. Chang, M. Chen, Y. Bao, and Z. Xu. Computer organization and design course with fpga cloud. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 927–933, 2019.

[67] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *2011 IEEE symposium on security and privacy*, pages 313–328. IEEE, 2011.

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

[69] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu. Hermes attack: Steal dnn models with lossless inference accuracy, 2020.