Stealing Neural Network Structure through Remote FPGA Side-channel Analysis

Yicheng Zhang, *Member, IEEE,* Rozhin Yasaei, *Member, IEEE,* Hao Chen, *Member, IEEE,* Zhou Li, *Senior Member, IEEE,* and Mohammad Abdullah Al Faruque, *Senior Member, IEEE*

***Abstract*—Deep Neural Network (DNN) models have been extensively developed by companies 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, which can significantly reduce her model development workload. We believe the threat presented by our attack is tangible, and new defense mechanisms should be developed against this threat.**

***Index Terms*—Deep neural network, cloud FPGA, side-channel analysis, hardware trojan.**

I. 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, natural language processing, and hardware application [1]–[3]. 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 [4], 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*. The power side-channel is shown to be effective for detecting malicious circuit manipulation [5], [6]. It also has been demonstrated that stealing model secret is feasible when launching CPU [7]–[9], or GPU [10]–[13] 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 [14], Google Compute Engine (GCE) [15], and Microsoft Azure [16], 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 [17]. 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 [18], 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 the 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 [19]), both layers and their hyper-parameters can be distinguished. We implemented our attack based on this observation on a ZedBoard [20] and evaluated end-to-end attacks on three DNN models: multiple layer perceptron(MLP) [21], AlexNet [22], and VGG16 [23]. 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 confidentiality of crypto cores or DNN models, they have shortcomings to address the following challenges in our attack scenario:

- Lacking physical access to the victim hardware platform makes it impossible to apply local side-channel attacks [24]–[27].
- The input and output data to the targeted model is not available to the adversary. Thus, conventional power analysis methods such as SPA [28] and DPA [18] would fail.
- The confidentiality of DNN depends on a broad set of elements (hyper-parameters and layers), which contains

a huge search space while crypto cores only have only one secret encryption key.

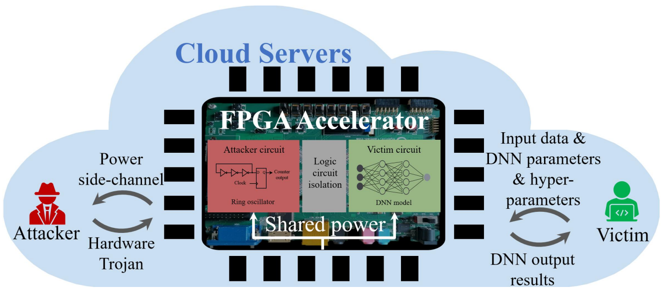

Fig. 1: Adversary model.

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

- The first attack to reconstruct the DNN model's architecture without physical access to the FPGA platform.
- The consideration of the power traces as time series and 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.

II. Background

A. 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. 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. Compared to the fixed-function Application Specific Integrated Circuit (ASIC), FPGAs can be programmed in less than a second and cost much less. Even though nowadays 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* [17] which supports space-sharing of FPGA instances among multiple tenants in the cloud.

Some prior works [29]–[32] present methodologies to the temporary allocation of FPGA instances by successively allocating the entire or part of the FPGA instance to tenants over time. Some research [33]–[37] 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' secrets in such an *FPGA multi-tenancy* scenario. Below we describe the setting and goal of the adversary.

B. 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 attempt 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. Prior work shows adversarial co-location [38], [39] is feasible, and Zhao et al. [18] 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 [20], which is also used by the prior FPGA remote attack [18]. Cloud providers utilize the FPGA board to serve FPGA users, like Xilinx Virtex Ultrascale+ used by Amazon F1 [14], 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 [36]), and it is difficult to run controlled attacks on production cloud without affecting other legitimate users. We discuss this limitation in Section V. Figure 1 illustrates the adversary model.

Before our work, Hua et al. carried out the model-stealing attack on FPGA [25]. 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.

DNN Structural Secret. DNN structure has been considered intellectual property for an AI company [4], 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 [9], the search space for VGG16, a broadly used DNN model, is 5.4×10^{12}, if the attacker guesses in a brute-force way.

Similar to prior works [9], [11], [12], [24], [25], [27], 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.
- 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 [9], [10], [24], 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 V.

III. Attack Methodology

A. FPGA Power Sensor

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 use it to infer the secret of the other victim. Zhao et al. [18] 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. 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 2, we illustrate the design of our RO sensor. For each power sensor, three inverters are connected sequentially along with an AND gate, and 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* [40], 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 (our experiment platform), we implement 20 RO sensors and connect their output to an adder tree to obtain their sum as the final power sensor value. The input signal of the first RO sensor is set to 1 while the other 19 RO sensors share the same enable signal. With this design, we can also make our attack stealthy, which profiles the victim DNN only when it is running. In particular, the first RO power sensor is activated all the time to monitor the power consumption. In contrast, the other 19 RO sensors are turned off initially by setting enable signal to be 0. When the adder tree returns abnormal readings, which means other users start to execute heavy workload, like DNN computation on the same board, the enable signal will be reset to be 1 to turn on all the other RO sensors. As a result, the adversary can identify the beginning of DNN execution. For each iteration of DNN execution, we collect over 20,000 samples from the FPGA board. To notice, the attacker can use other trojan designs that are alternative to Figure 2, and we discuss their performance in Section IV-G. In Section IV, we discuss the impact caused by the number of RO sensors.

[1]On FPGA cloud instances like Amazon F1, PCIe Physical Functions (PFs) are provided to tenants for accessing and controlling their FPGA logic [14].

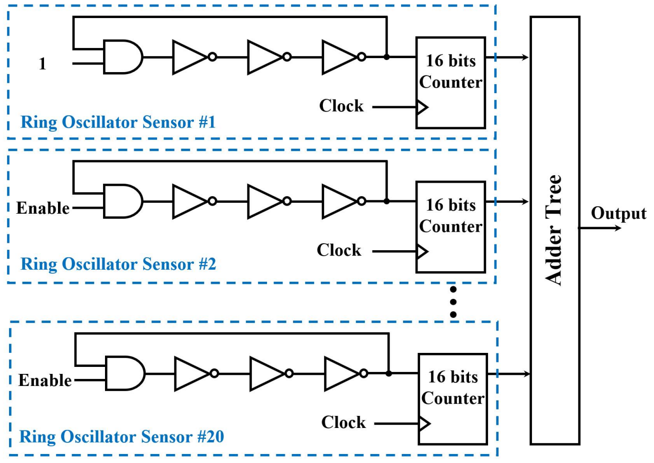

Fig. 2: Ring oscillator power sensors.

Placement of RO power sensors. How RO sensors are placed on FPGA impacts the attacker's readings, further influencing the attack accuracy. Because FPGA logic of different users can share the same FPGA board, cloud providers enforce placement policies for every user. Figure 3(a) and Figure 3(b) illustrate the floorplan related to different policies when one user is an attacker: free placement and restricted placement. In Section IV-G, we evaluate how the placement policy impacts attack accuracy.

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(a), 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.

B. DNN Layers and Computational Workload

Deep neural networks are a family of methods that transfer input to output with a variety of repetitious processing units. Each processing unit is called a *layer*. The full set of layers, including their hyper-parameters settings, are named DNN *structure*. For a power sensor to discern structural elements

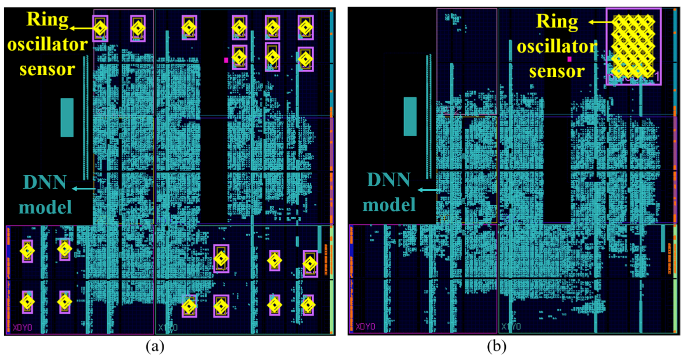

Fig. 3: The floorplan of FPGA (a) when the attacker has permission to determine the placement of RO and (b) without permission to determine the locations of RO.

of a DNN model, each element should incur a different computation workload. Below we analyze the three most popular layers' workload, namely fully-connected, convolutional, and pooling layers. Each layer is also associated with a set of hyper-parameters, and we list them in Table I. The equations about the workload analysis are listed in Equation 1 to 4 at the end of this subsection. Equation 5 lists an important constraint about hyper-parameters, which is leveraged for result correction, as described in Section III-C.

TABLE I: Hyper-Parameter 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

• **Fully-connected (FC) layer.** A fully-connected layer consists of a number of neurons, and 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 1.

• **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 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 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]. Except for W_i, all other inputs are hyper-parameters set by the model developer. W_i can be computed through Equation 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 2, except that MAC is replaced by MAX and W_i is replaced by $W_{pooling}$ (see Equation 4).

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

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

$$W_i = \frac{W_{i-1} - F + 2 \times P}{S} + 1 \tag{3}$$

$$W_{pooling} = \frac{W_{i-1} - F}{S} + 1 \tag{4}$$

$$S \leq F \leq \frac{W_i}{2} \tag{5}$$

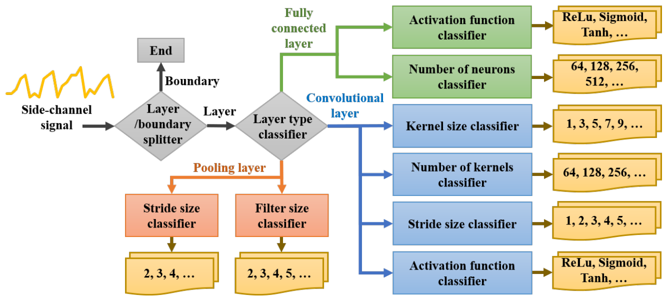

Fig. 4: The attack workflow to infer DNN structure.

C. 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.

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

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 4, and we elaborate on each stage below.

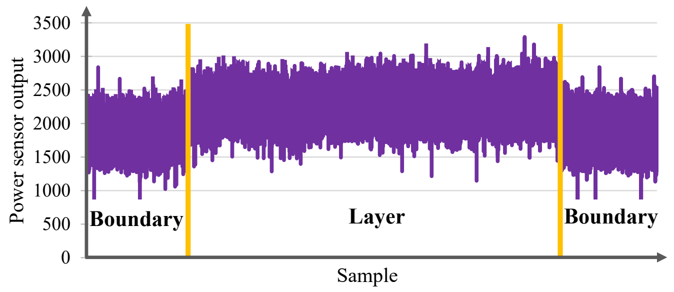

Fig. 5: The output signal of RO sensors under convolutional layer.

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 5 illustrates the readings of *"boundary"* and *"layer"*. After this step, the number of layers would also be revealed.

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 1 and 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.

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 1, 2, and 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.

IV. Evaluation

A. Experiment Setup

We run experiments on a ZedBoard [20], which contains a dual Corex-A9 Processing System (PS) with 85,000 Series-7 Programmable Logic (PL) cells. It also has 21 DSPs, 177334 FFs, 20607 LUTs, and 235 BRAMs. Similar to Hua et al. [25], the layers are executed sequentially. For most of the tasks, we deploy 20 RO power sensors and follow the floorplan shown in Figure 3(a). For the RO power sensor's implementation, we utilize the typical FPGA design flow provided by Xilinx Vivado, and all components of RO sensors are written in Verilog, which is 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.

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. For example, the ZFNet [41], which is the champion of ImageNet 2013 Challenge, has the same layer types as AlexNet but improved it by adopting new hyper-parameters. For the convolutional layer, the filter size is set to be an odd number ranging from 1 to 13, the size of stride ranging varying 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 [42] (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. For each hyper-parameter value, we run AlexNet on the FPGA for 50 iterations, 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). The victim DNN model is trained with the ImageNet dataset on an Nvidia GTX 1080 GPU to derive the model weights. Then, the victim models run under the *testing* mode (only feed-forward) on the FPGA board. Each model is executed 20 times with the same input, and we aggregate the accuracy across models. For each trace of iteration, we collect over 20,000 samples from the FPGA board. For the final end-to-end analysis, we report the accuracy for each tested model individually. Though we only simulate the victim model in the testing model, inferring model secret

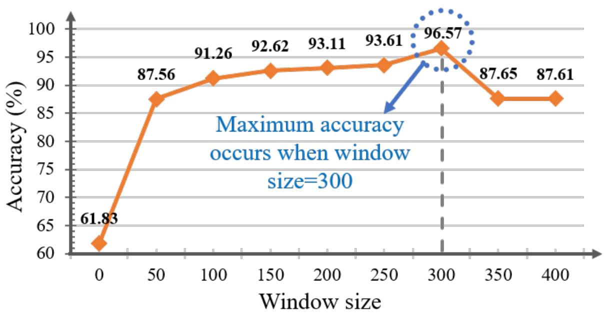

Fig. 6: The accuracy of layer segmentation for different window sizes.

TABLE II: Selected Feature (top 5).

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

when DNN is *training* is also feasible, as training only adds a back-propagation process after a batch of input data goes through all layers.

B. Layer Segmentation

We use XGBoost [19], 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 6, 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 as related to the layer boundary and the remaining ones related to the layers.

C. 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` [43], 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 II lists the top 5 features.

D. 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 III (Column "Layer type"). 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, and 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 III. Among all classifiers, tree-based models like XGBoost and RandomForest perform best, reaching **100%** accuracy in this task.

E. 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 III. 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 for the inference of 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).

F. End-to-end Result

Finally, we assemble the inferred layers and their hyper-parameters together 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 IV. As each model is tested for 20 traces, for the predicted sequence, we show the optimal one, 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

TABLE III: 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 [44]	$n\_neighbors = 5$	87.5	35	37.14	37.14	86.67	100
Gradient Boosting Classifier [45]	$n\_estimators = 1000$	97.5	**96.67**	91.42	71.42	96.67	96.67
Decision Tree [46]	-	97.5	94.67	68.57	68.57	96.67	100
Random Forest [47]	$n\_estimators = 1000$	**100**	71.67	57.14	31.42	93.33	100
Multi-layer Perceptron classifier [48]	$activation\_function = relu, hidden\_layers = 1000, solver = adam, max\_iter = 1000$	90	15	14.28	20	73.33	90
Naive Bayes [49]	-	87.5	23.33	17.14	20	90	90
AdaBoost [50]	-	97.5	40	31.42	31.42	**97.47**	100
XGBClassifier [19]	$n\_estimators = 1000, max\_depth = 5$	**100**	93.33	**94.28**	**94.85**	96.67	**100**

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

G. Impact of RO Power Sensor Configurations

Number of ROs. We change the number of ROs from 1 to 20 to assess its impact on the attack accuracy. For simplicity, we only show the result of layer segmentation in Figure 7, 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.

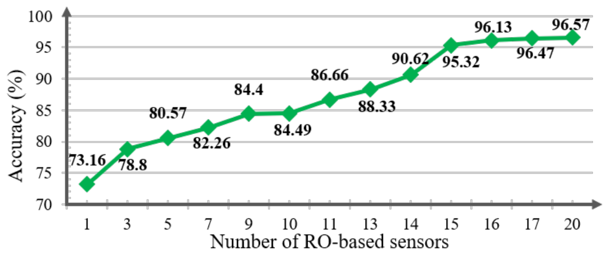

Fig. 7: The accuracy of layer segmentation for different number of power sensors.

Placement of ROs. We consider the two placement policies for RO power sensors. As Figure 3(a) shown, in this situation, the attacker can constrain the placement of each RO power sensor wherever she wants. The second scenario, like Figure 3(b) 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(b) 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 the sum of 20 RO power sensors readings, making variations in individual RO negligible.

H. Alternative RO Power Sensors

The power sensor we use for the attack is based on the conventional RO sensor, which has also been used by the prior work for remote FPGA attack [18]. On the other hand, we found there are cloud providers examining the combinatorial loops in the logic before it is instantiated on the FPGA board [51]. The adversary would need to change the design of the sensor to avoid detection. Specifically, we found there exist two alternative designs, Latch-based RO sensor [52] and Flip-Flop based RO sensor [53], that is capable of achieving a similar sampling resolution as the conventional RO sensor. The architectures of these two ROs are shown in Figure 8(a) and Figure 8(b), which replaces an inverting buffer with a latch and a flip-flop, respectively. 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 III-C. We retrained 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 [54] can also be exploited as a power sensor. But as studied by Zhao et al. [18], compared to the RO sensor, it is less stable and

TABLE IV: 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%

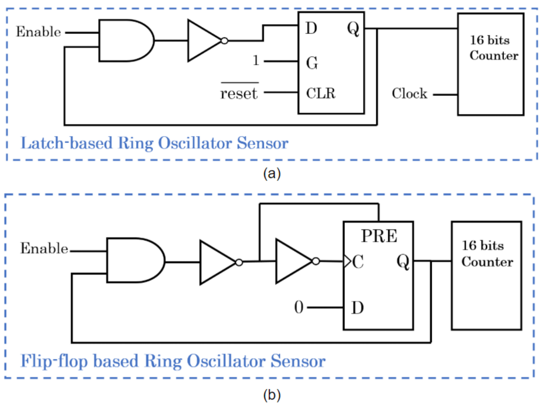
Enable
D
Q
1
G
reset
CLR
Clock
16 bits Counter
Latch-based Ring Oscillator Sensor
(a)
Enable
PRE
C
Q
0
D
16 bits Counter
Flip-flop based Ring Oscillator Sensor
(b)

Fig. 8: Two design of RO sensors a) Latch-based and (b) Flip-flop based.

has lower power resolution. Therefore, we choose RO sensors as our power trojan for most of the evaluation tasks.

I. Effectiveness of the Recovered Models

While we successfully recovered the layer structure and hyper-parameters for each layer at high accuracy, we have not discussed how much help it gives to the attacker in building her own model. It is theoretically possible that even when the attacker recovered over 90% hyper-parameters of the target DNN, the recovered model could only reach 50% accuracy due to reasons like the missing of key hyper-parameters. As a result, we try to measure the value of the stolen model by comparing its classification accuracy to the original model. Specifically, we programmed the original and recovered version of AlexNet using the layer sequences shown in Table IV) with pytorch-V1.5 and NVidia CUDA-V10.2. The models are trained and tested on Tiny imageNet-200 dataset [55], 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 comparing to 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 [56] 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 V, suggesting the adversary's workload in model tuning will be significantly reduced when starting from the stolen model.

TABLE V: Accuracy of original and recovered model.

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

By exploiting the inferred model structure, the attacker can also launch an adversarial attack [4] such as evasion more effectively. Hu et al. [10] 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.

J. Signal-to-Noise Ratio (SNR)

Signal-to-noise ratio (SNR), introduced by Mangard et al. [57], is a common metric to measure the quality of a side-channel. Here, we compute the SNR of our power-based side-channel and measure how it varies when the environment is not ideal for the attacker, e.g., another irrelevant task runs on the same FPGA board. The equation below computes SNR.

$$SNR = \frac{Var_{(Signal)}}{Var_{(Noise)}} \qquad (6)$$

Where $Var_{(Signal)}$ is the variance of the RO power sensor readings when the DNN model is executed, and $Var_{(Noise)}$ is the variance of readings when DNN is not executed.

In particular, we run AlexNet for 5 iterations and collect 80,000 samples to compute the average SNR. When DNN execution is the only active task, the measured SNR is 2.35, suggesting the quality of our side-channel is satisfactory for model stealing.

When another irrelevant task is running, e.g., by another user sharing the board, the SNR is expected to be decreased since more noises from the other task will be observed by the RO sensors. To measure the impact, we write another program, which runs floating-point matrix multiplication, and places it together with AlexNet. They are separated logically and physically under Vivado Pblock constraint. It turns out the SNR decreases to 1.73 from 2.35. Still, this value is much larger than 1. Then, we measure how the inference accuracy is impacted. We rerun the layer/boundary classifiers with the additional noise generated by the other program. The accuracy of this task drops to 89.36% from 96.57% when DNN is the only active task. Though the drop is noticeable, we expect the attacker still has a good chance of guessing the right model structure.

K. Parallel Execution of DNN

Our attacks are shown to be effective when the model is executed sequentially. Yet, executing DNN in parallel is possible on FPGA, at four different levels, i.e., Task-level (multiple prediction tasks are executed simultaneously), Layer-level (different layers are executed at the same time for different inputs), Loop-level (multiple kernels of a layer are executed simultaneously) and Operator-level (multiple MACs are performed simultaneously) [58]. We examined the Task-level parallelism and found our attack is still effective. In particular, we launch 5 image prediction tasks simultaneously on the same FPGA board and capture the aggregated power traces with the RO sensors. We retrained layer/boundary classifiers using the new RO sensor readings, grouped into windows of 300 data points. In the end, the accuracy on layer segmentation can reach 95.42%, which is slightly lower than the case of sequential execution (96.57%). Regarding other types of parallelisms, maintain the same level of accuracy is more challenging due to the large search space of the combination between layers/kernels/MACs. We leave the exploration on the other types as future work.

V. Discussion

A. Limitation

While our attack is shown to be effective under the *FPGA multi-tenancy* scenario, main cloud FPGA providers only support *FPGA single-tenancy* for now. Yet, we argue that the attack should be mitigated as multi-tenant FPGA is expected to be adopted in the near future [59]. Alternatively, we can explore *cross-FPGA* side-channels to steal the model. In fact, very recently, Giechaskiel et al. [60] showed the power supply unit (PSU) could be exploited to construct FPGA-to-FPGA, CPU-to-FPGA, and GPU-to-FPGA covert channels between different boards. On top of these side-channels, model-stealing might be possible on the current commercial FPGA cloud, and we leave this exploration as future work.

To avoid impacting legitimate users of the FPGA cloud, our experiment is executed on a Xilinx Zynq-7000 FPGA board, Zedboard [20] locally, which is also used by the prior works launching FPGA remote attack [18]. Our attack is expected to apply to the cloud, which uses the FPGA board to serve clients, like Amazon F1 [14].

Our attack focuses on DNN structure, leaving neuron weights not inferred. Hua et al. [25] showed weights could be inferred on CNN accelerators, assuming "zero pruning" is done on the feature map of the CNN model. Multiple works [24], [61], [62] based on DPA were able to infer DNN weights on CNN accelerators or FPGA boards. However, they all assume the adversary can feed input to the device, which is often requires the adversary to have physical control, which is not allowed under remote attacks. One might think weight inference might be easy when a simple DNN model is examined. However, our preliminary result suggests that is not the case: we tried to infer the weights of a 3-layer (3-2-2) MLP that is executed as a Binarized Neural Network (BNN) [63] with random inputs, but the accuracy is less than 50% with the XGBoost model. Alternatively, we can relax the restriction on the attacker and assume part of the input data is public. We leave this exploration as future work.

B. Countermeasures

Our attack could be thwarted by two countermeasures potentially: hiding the power consumption patterns unique to the DNN layers/hyper-parameters or rejecting the deployment request of suspicious FPGA logic. The first countermeasure schema requires supplementary hardware [64] to be installed to mask the power consumption (e.g., through noise injection) of every operation or embed active fences [65] 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 or netlist file. This countermeasure requires the provider to have a "blacklist" about the logic to be rejected [51]. In recent years, multiple works proposed electrical-level [66] scanning methods to prevent malicious logic implementation on Multi-Tenant FPGAs. Gnad et al. [67] analyze FPGA bitstreams to detect malicious logic [66]–[69]. Krautter et al. [66] develop an electrical-level bitstream checking methodology to detect malicious logic on multi-tenant FPGAs. La et al. [68] demonstrate malicious FPGA logic can be rejected by scanning a malicious construct in the netlist. Ahmed et al. [69] present a bitstream-level proof-carrying hardware approach to detect the stealthy malicious logic. However, these defense methods require regular renewal of the "blacklist" that characterizes malicious logic. As such, using alternative RO designs (e.g., latch-based [52] or flip-flop-based [53] RO circuits introduced in Section IV-H) might help the attacker bypass the checks and achieve similar attack accuracy. For instance, Ahmed et al. [70] show that by injecting trojan in a post-synthesis step and making it unconnected in the bitstream, their attack can circumvent bitstream-level verification. So far, we have not found a bullet-proof defense approach against the power trojans. In other words, a solution that can provably mitigate our attack without incurring high overhead seems non-trivial.

VI. Related work

As the machine-learning model becomes intellectual property, A number of prior works [7]–[13], [24], [25], [27], [61],

[71]–[74], [74], [75] 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 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) [72]–[75]. Recently, a few works examined the API-based model stealing on DNN models and it turns out when leveraging adversarial examples [74] or functionally-equivalent extraction [71]. In the second direction, prior works demonstrated the possibility to use cache side-channel analysis, EM analysis, and power analysis on CPU [7]–[9], GPU [10]–[13], microcontroller [24] and FPGA [25], [27], [61] to infer the model secret. The research closest to ours was done by Hua et al. [25], and Yu et al. [27]. 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)**.

Our attack can be categorized as a specific type of remote FPGA attack [18], [52], [54], [76]–[80]. 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) have been shown to be powerful as voltage- and temperature-based sensors in attacking the shared FPGA. Giechaskiel et al. [52], [76] and Provelengios et al. [77] 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. [18] performed a SPA (simple power analysis) attack on an RSA crypto module with an RO-based power sensor. Similarly, Schellenberg et al. [54] reconstructed an AES encryption module by TDCs instead of ROs on a Spartan-6 FPGA. Tian et al. [78] showed that it is feasible to create thermal covert channels in the real cloud FPGAs by using ROs. Shayan et al. [79] demonstrated that TDCs (Time-to-Digital Converters) could be utilized to recover the input images on share FPGA instances. Moini et al. [80] adopt a similar threat model as ours while exploiting FPGA power side-channel information collected by TDCs to recover the input images. To the best of our knowledge, we are first to 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 [81]. More recently, Giechaskiel et al. [60] showed the power supply unit (PSU) can be exploited to construct covert channels across boards.

VII. Conclusion

In this paper, we investigate 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 attention of the security community to develop new cost-effective defense solutions against this threat, and we plan to investigate it as well.

References

[1] R. Yasaei, F. Hernandez, and M. A. Al Faruque, "Iot-cad: context-aware adaptive anomaly detection in iot systems through sensor association," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[2] R. Yasaei, S.-Y. Yu, and M. A. A. Faruque, "Gnn4ip: Graph neural network for hardware intellectual property piracy detection," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2021.

[3] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque, "Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2021, pp. 1504–1509.

[4] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *IEEE Eur. Symp. Secur. and Privacy (EuroS&P)*, 2018, pp. 399–414.

[5] S. Faezi, R. Yasaei, and M. Al Faruque, "Htnet: Transfer learning for golden chip-free hardware trojan detection," *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2021.

[6] S. Faezi, R. Yasaei, A. Barua, and M. A. Al Faruque, "Brain-inspired golden chip free hardware trojan detection," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2697–2708, 2021.

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

[8] S. Hong *et al.*, "Security analysis of deep neural networks operating in the presence of cache side-channel attacks," *arXiv preprint arXiv:1810.03487*, 2018.

[9] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn dnn architectures," in *Proc. USENIX Secur. Symp.*, 2020, pp. 2003–2020.

[10] 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 *Proc. Arch. Sup. for Progr. Lang. and Op. Sys. (ASPLOS)*, 2020.

[11] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2018, pp. 2139–2153.

[12] 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 *Proc. IEEE/IFIP Int. Conf. on Depen. Sys. and Net. (DSN)*, 2020, pp. 125–137.

[13] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes attack: Steal dnn models with lossless inference accuracy," in *Proc. USENIX Secur. Symp.*, 2021.

[14] Amazon, "Amazon ec2 f1 instances," https://aws.amazon.com/ec2/instance-types/f1/.

[15] Google, "Google compute engine(gce)," https://cloud.google.com/compute/.

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

[17] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on fpga virtualization," in *Proc. IEEE Int. Conf. Field-Progr. Logic and Ap. (FPL)*, 2018, pp. 131–1317.

[18] M. Zhao and G. E. Suh, "Fpga-based remote power side-channel attacks," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2018, pp. 229–244.

[19] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. ACM SIGKDD Conf. on Knwl. Discov. and Data Min. (KDD)*, 2016, pp. 785–794.

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

[21] R. Lippmann, "An introduction to computing with neural nets," *IEEE Assp magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

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

[24] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Csi nn: reverse engineering of neural network architectures through electromagnetic side channel," in *Proc. USENIX Secur. Symp.*, 2019.

[25] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Design Autom. Conf. (DAC)*, 2018, pp. 1–6.

[26] 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 *Proc. Annu. Comput. Secur. Ap. Conf. (ACSAC)*, 2018, pp. 393–406.

[27] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "Deepem: Deep neural networks model recovery through em side-channel information leakage," in *Proc. IEEE Int. Symp. on Hw. Or. Secur. and Trust (HOST)*, 2020, pp. 209–218.

[28] S. Mangard, "A simple power-analysis (spa) attack on implementations of the aes key expansion," in *Int. Conf. on Inf. Secur. and Cryptol.* Springer, 2002, pp. 343–358.

[29] A. A. Al-Aghbari and M. E. Elrabaa, "Cloud-based fpga custom computing machines for streaming applications," *IEEE Access*, vol. 7, pp. 38 009–38 019, 2019.

[30] G. Dai, Y. Shan, F. Chen, Y. Wang, K. Wang, and H. Yang, "Online scheduling for fpga computation in the cloud," in *Proc. IEEE Int. Conf. on Field-Progr. Tech. (FPT)*, 2014, pp. 330–333.

[31] 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*. Springer, 2019, pp. 9–33.

[32] K. Zhang, Y. Chang, M. Chen, Y. Bao, and Z. Xu, "Computer organization and design course with fpga cloud," in *Proc. ACM Techn. Symp. on Comput. Sci. Edu.*, 2019, pp. 927–933.

[33] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with openstack," in *Proc. IEEE Int. Symp. on Field-Progr. Cust. Comp. Mach. (FCCM)*, 2014, pp. 109–116.

[34] 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*, 2014, pp. 1–10.

[35] J. M. Mbongue, F. Hategekimana, D. T. Kwadjo, and C. Bobda, "Fpga virtualization in cloud-based infrastructures over virtio," in *Proc. IEEE Int. Conf. on Comp. Design (ICCD)*, 2018, pp. 242–245.

[36] J. M. Mbongue, A. Shuping, P. Bhowmik, and C. Bobda, "Architecture support for fpga multi-tenancy in the cloud," in *Proc. IEEE Int. Conf. on App.-specific Sys., Archi. and Process. (ASAP)*, 2020, pp. 125–132.

[37] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling fpgas in hyperscale data centers," in *Proc. IEEE 12th Int. Conf. on Ubi. Intel. and Comp. and IEEE 12th Int. Conf. on Auto. and Trust. Comp. and IEEE 15th Int. Conf. on Scal. Comp. and Comm. and Its Asso. Work. (UIC-ATC-ScalCom)*, 2015, pp. 1078–1086.

[38] 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 *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2009.

[39] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2011.

[40] Xillybus, "Xillybus product brief," http://xillybus.com/downloads/xillybus_product_brief.pdf.

[41] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Eur. Conf. on Comp. Vis. (ECCV)*. Springer, 2014, pp. 818–833.

[42] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. on Comp. Vis. and Pat. Recog. (CVPR)*, 2009.

[43] 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*, 2018.

[44] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *Int. Joint Conf. on Comp. Vis.-, Img. and Com. Graph. Theory and App. (VISAPP)*, vol. 2, no. 331-340, p. 2, 2009.

[45] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[46] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. on Sys., Man, and Cyber.*, vol. 21, no. 3, pp. 660–674, 1991.

[47] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[48] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journ. of Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[49] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[50] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

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

[52] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Reading between the dies: Cross-slr covert channels on multi-tenant cloud fpgas," in *Proc. IEEE Int. Conf. on Comp. Design (ICCD)*, 2019, pp. 1–10.

[53] 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*, 2019.

[54] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on fpgas," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2018, pp. 1111–1116.

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

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

[57] S. Mangard, "Hardware countermeasures against dpa–a statistical analysis of their effectiveness," in *Cryptographers' Track at the RSA Conference*. Springer, 2004, pp. 222–235.

[58] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-optimized fpga accelerator for deep convolutional neural networks," *ACM Trans. on Reconfig. Tech. and Sys. (TRETS)*, vol. 10, no. 3, pp. 1–23, 2017.

[59] R. Elnaggar, R. Karri, and K. Chakrabarty, "Multi-tenant fpga-based reconfigurable systems: Attacks and defenses," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2019.

[60] K. Rasmussen, I. Giechaskiel, and J. Szefer, "Capsule: Cross-fpga covert-channel attacks through power supply unit leakage," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2020, pp. 1728–1741.

[61] K. Yoshida, T. Kubota, M. Shiozaki, and T. Fujino, "Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis," in *Proc. IEEE Int. Symp. on Field-Progr. Cust. Comp. Mach. (FCCM)*, 2019, pp. 318–318.

[62] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *Proc. IEEE Int. Symp. on Hw. Or. Secur. and Trust (HOST)*, 2020, pp. 197–208.

[63] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[64] T. Popp, S. Mangard, and E. Oswald, "Power analysis attacks and countermeasures," *IEEE Design & test of Computers*, vol. 24, no. 6, pp. 535–543, 2007.

[65] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active fences against voltage-based side channels in multi-tenant fpgas," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[66] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant fpgas in the cloud," *ACM Trans. on Reconfig. Tech. and Sys. (TRETS)*, vol. 12, no. 3, pp. 1–26, 2019.

[67] D. R. Gnad, S. Rapp, J. Krautter, and M. B. Tahoori, "Checking for electrical level security threats in bitstreams for multi-tenant fpgas," in *Proc. IEEE Int. Conf. on Field-Progr. Tech. (FPT)*. IEEE, 2018, pp. 286–289.

[68] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "Fpgadefender: Malicious self-oscillator scanning for xilinx ultrascale+ fpgas," *ACM Trans. on Reconfig. Tech. and Sys. (TRETS)*, vol. 13, no. 3, pp. 1–31, 2020.

[69] Q. A. Ahmed, T. Wiersema, and M. Platzner, "Proof-carrying hardware versus the stealthy malicious lut hardware trojan," *Applied Reconfigurable Computing*, vol. 11444, 2019.

[70] Q. A. Ahmed *et al.*, "Malicious routing: Circumventing bitstream-level verification for fpgas," 2021.

[71] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *Proc. USENIX Secur. Symp.*, 2016, pp. 601–618.

[72] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proc. USENIX Secur. Symp.*, 2016, pp. 601–618.

[73] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2018.

[74] 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 *Net. and Distrib. Sys. Secur. Symp. (NDSS)*, 2020.

[75] X. Yuan, L. Ding, L. Zhang, X. Li, and D. Wu, "Es attack: Model stealing against deep neural networks without data hurdles," *arXiv preprint arXiv:2009.09560*, 2020.

[76] I. Giechaskiel, K. Eguro, and K. B. Rasmussen, "Leakier wires: Exploiting fpga long wires for covert-and side-channel attacks," *ACM Trans. on Reconfig. Tech. and Sys. (TRETS)*, vol. 12, no. 3, pp. 1–29, 2019.

[77] 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 *Proc. ACM/SIGDA Int. Symp. on Field-Progr. Gate Arrays (FPGA)*, 2019, pp. 292–297.

[78] S. Tian and J. Szefer, "Temporal thermal covert channels in cloud fpgas," in *Proc. ACM/SIGDA Int. Symp. on Field-Progr. Gate Arrays (FPGA)*, 2019, pp. 298–303.

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

[80] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Power side-channel attacks on bnn accelerators in remote fpgas," *IEEE Journ. on Emerg. and Sel. Topics in Circ. and Sys.*, 2021.

[81] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "Remote inter-chip power analysis side-channel attacks at board-level," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–7.