A Side-channel Power Attack on Machine Learning Models in FPGAs

Zixuan Fang
Columbia University
New York, USA
zf2324@columbia.edu

Weitao Lu
Columbia University
New York, USA
wl2928@columbia.edu

Zhiwei Xie
Columbia University
New York, USA
zx2468@columbia.edu

ABSTRACT

Cloud FPGA services are playing an increasingly important role in cloud computing, providing efficient hardware support for various acceleration algorithms and holding commercial value. However, in the multi-tenant environment of cloud FPGAs, sharing hardware resources also brings potential security risks, where attackers may conduct side-channel attacks and other methods to steal sensitive information from victims residing on the same cloud FPGA. This paper mainly investigates how to obtain and clone machine learning models deployed on FPGAs through voltage-based side-channel attacks and reverse engineering. We first build three different types of machine learning models (three-layer neural network, convolutional neural network, and ResNet-50) using Python and deploy them on FPGA boards using the hls4ml tool. Then, we use the XADC to monitor the voltage fluctuations of the FPGA chip during runtime and collect voltage side-channel information. Based on this information, we employ reverse engineering techniques to train a cloned model functionally equivalent to the victim's model and compile it into a bitstream, ultimately deploying it on other FPGAs. Through this work, we unveil the security risks faced by machine learning models in the cloud FPGA multi-tenant environment, provide optimization strategies for deploying machine learning models on FPGAs, and finally discuss potential defense strategies.

KEYWORDS

FPGA, side-channel analysis, machine learning models, reverse engineering

ACM Reference Format:
Zixuan Fang, Weitao Lu, and Zhiwei Xie. 2024. A Side-channel Power Attack on Machine Learning Models in FPGAs . In *COMS6424, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

With the continuous development of machine learning and artificial intelligence algorithms, the demand for hardware acceleration is increasing day by day. FPGA (Field-Programmable Gate Array), as a reconfigurable hardware accelerator, is widely used in cloud computing and data centers, providing efficient acceleration for various algorithms. Cloud providers integrate FPGA resources as cloud services and offer them to users in a multi-tenant mode, which can greatly improve the hardware utilization rate [11], reduce costs, and meet diverse needs. Therefore, cloud FPGA services have tremendous commercial value and development potential.

However, the FPGA multi-tenant environment also brings some security risks. First, since multiple tenants share the same hardware platform, there is a potential risk of information leakage. Attackers may obtain sensitive information or model parameters of other tenants through side-channel attacks and other means. Second, to simplify management, cloud providers often design standardized interface structures and protocols, making all tenants' operations follow uniform specifications. This standardized operation provides an opportunity for attackers to infer the victim's behavior on the same FPGA [17]. Additionally, attackers may exploit physical attacks, reverse engineering, and other means to compromise the security of FPGAs. Currently, research has reported various attack methods, such as power-based attacks [13] and temperature analysis [20].

In this research, we focus on the attack method that combines Voltage-based Side-channel Attack and Reverse Engineering via Voltage Monitoring. By monitoring the voltage fluctuations of the FPGA chip, attackers can infer the structure and parameters of the machine learning model deployed on the FPGA, thereby achieving the theft and cloning of the model. We first use Python to build and train the victim's machine learning model, and optimize the deployed model through techniques such as pruning to conserve more resources on the FPGA. Then, we utilize the hls4ml tool to convert the model into a High-Level Synthesis language, and further translate it into a Hardware Description Language (HDL), generating a bitstream, and finally deploy it on other FPGA boards. During the model's execution, we monitor the voltage changes of the chip using the XADC (Xilinx Analog-to-Digital Converter) on the FPGA. Based on the collected voltage side-channel information, we use reverse engineering techniques to obtain a cloned model functionally equivalent to the victim's model through training. Finally, we compile the cloned model into a bitstream and can deploy it on other FPGAs, achieving the theft and replication of the victim's model.

Our original plan was to investigate attacks targeting cloud FPGAs. However, due to limitations in the experimental conditions, we decided to use physical FPGA boards as an alternative. Although physical FPGAs differ from cloud FPGAs in terms of deployment and access methods, they share similarities in hardware architecture and working principles. Therefore, the attack experiments and analyses conducted on physical FPGAs can still provide meaningful references and insights for the security research of cloud FPGAs.

Conference acronym 'XX, May 05,2024, New York, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 BACKGROUND

This section provides background on the FPGA and three machine learning models used for evaluation.

2.1 PYNQ-Z2

This work introduces the PYNQ-Z2 as the hardware platform for our study. As an open-source project by AMD, it offers extensive documentation and community support, simplifying the development process and reducing complexity. Furthermore, compared to other FPGA and microprocessor platforms, the PYNQ-Z2 provides shorter inference times and lower power consumption when running the same machine learning models, reducing testing time and making it an ideal choice for deploying machine learning models and conducting security research.

2.2 Three Models Used for Evaluation

To assess the performance and security of deploying machine learning models on FPGAs, we selected three different models for experimentation to investigate how models of varying complexities perform under specific attacks.

2.2.1 Neural Network. Neural networks are a milestone in the field of machine learning, simulating the working principles of biological neurons to handle complex nonlinear problems. They have achieved significant success in areas such as pattern recognition and natural language processing and are widely used in industrial and academic research. In this study, we chose neural networks as one of the targets for attacks because they form the basis of many advanced models, and understanding their security is crucial to protect more complex models.

The concept of the neural network was first proposed by Warren McCulloch and Walter Pitts in 1943 [12], followed by significant developments such as the perceptron in 1958 [15], and the backpropagation algorithm in 1986 [16]. A typical neural network consists of an input layer, hidden layers, and an output layer. The input layer receives the raw data, the hidden layers perform nonlinear transformations, and the output layer produces the final results. Each layer is made up of multiple neurons, which are interconnected by weights. The layers interact and update parameters through forward and backward propagation. Forward propagation passes input data to the output layer, while backward propagation optimizes the weights based on the loss function.

The computational processes in the layers of a neural network can be affected by voltage-based side-channel attacks and reverse engineering via voltage monitoring, leading to the leakage of sensitive information or the theft of the model. The advantage of neural networks lies in their ability to automatically learn features and patterns in the data, providing strong nonlinear expressive capabilities. Common neural network architectures include feedforward neural networks[24] and recurrent neural networks (RNNs) [5].

2.2.2 Convolutional Neural Networks (CNNs). CNN is a type of neural network specifically designed to process grid topology data, such as images. It has achieved tremendous success in computer vision tasks like image classification and object detection, and is widely used in the industry. In this study, we selected CNN as one of the targets for attacks because it represents an important class of neural network structures, which attackers might try to exploit to steal model parameters or infer sensitive information.

The concept of CNN was first proposed by Kunihiko Fukushima in 1980 [2], followed by Yann LeCun's introduction of the famous LeNet architecture in 1998 [9]. CNNs typically consist of convolutional layers, pooling layers, and fully connected layers. The convolutional layers perform convolution operations on input data to extract local features; pooling layers downsample feature maps to reduce data dimensions; fully connected layers classify or regress the features. Convolutional and pooling layers reduce the number of parameters through local connections and weight sharing, enabling CNNs to efficiently handle high-dimensional data. The fully connected layers map the extracted features to the output space.

The advantage of CNNs lies in their ability to extract local features from data, their translational invariance, and relatively fewer parameters. Famous CNN architectures include LeNet [9], AlexNet [8], and ResNet [3]. For this study, the multilayer and complex structure of CNNs provide detailed data processing insights, which help in studying how information might be leaked through side channels.

2.2.3 ResNet. ResNet is a deep neural network that addresses the difficulty of training deep models by introducing residual connections. It has achieved outstanding performance in tasks such as image classification and object detection, and is widely used in academic research and industrial deployment. In this study, we chose ResNet as one of the targets for attacks because it represents the current trend in deep learning, and attackers may be interested in such complex models.

ResNet was proposed by Kaiming He and others in 2015 [3] and has since become a foundational model for many computer vision tasks. ResNet consists of multiple residual blocks, each containing two or three convolutional layers, typically followed by batch normalization and ReLU activation functions. The key component of the residual block is the skip connection, which can be an identity connection or include convolutional operations to match the dimensions of the input and output. This design allows gradients to flow directly to shallower layers, alleviating the problem of vanishing gradients and enabling the network to more easily learn identity mappings. The stacking of such residual blocks forms the complete ResNet model.

For this study, ResNet provides a rich context for analyzing complex side-channel attacks and voltage monitoring. The advantage of ResNet lies in its ability to train very deep neural networks and its good generalization capabilities. Common ResNet architectures include ResNet-18, ResNet-50, ResNet-101, etc., with the numbers indicating the layers of the network.

2.3 Threat Model

When assuming the attack scenario, we still consider cloud FPGAs as the target environment. Specifically, we assume that the victim is a tenant who deploys machine learning models on a cloud FPGA, and these models are the victim's core intellectual property. The attacker's goal is to steal these models for economic benefits or other malicious purposes.

We assume that the attacker and the victim reside on the same cloud FPGA instance but are logically isolated from each other.

Under these circumstances, the attacker can collect side-channel information by observing voltage fluctuations of the shared FPGA instance and carry out attacks accordingly.

3 DEPLOY MACHINE LEARNING MODELS TO FPGAS

Machine learning models are complex and hard to write directly in a hardware description language. In that case, we will first build and train our model in python, then transfer the model to a High-level synthesis language. For this work we used C, and at last, translated it to HDL, generated the bitstream, and applied it to a board. We used hls4ml API to do the job, we also did some model optimization like pruning to make the machine learning model use fewer resources on the FPGA. At last, we trained three different models to evaluate our experiment.

3.1 hls4ml

hls4ml[1] can provide an translation of machine learning models from open-source packages (like Keras and PyTorch) for training machine learning algorithms to high level synthesis (HLS) code that can then be transpiled to run on an FPGA. The resulting HLS project can be then used to produce an IP which can be plugged into more complex designs or be used to create a kernel for CPU co-processing.

3.2 Model Optimization

The FPGA has several kinds of resources that will be used greatly in the building and running of the machine learning model. One of the most used resources is LUT, in order to successfully applied our model to the FPGA board, we need to optimization the structure of the model, the weights and how they are stored, we discussed three ways to do it.

Prune dense and convolutional layers can be used to reduce the number of non-zero weights. Sparse models generally require fewer computational resources for executing forward and backward passes because many operations involve zero values that can be skipped during computation. This enables the model to perform faster, especially on resource-constrained devices. Besides, Sparse models have more zeros in their weights and require less memory to store. This is particularly important for FPGA. In our work we tried to find a balance between the sparsity and the accuracy of our model. For the 3 layer network we set the sparsity to be 75% and for our convolutional network is 50%.

Quantization is a technique to reduce the computational and memory costs of running inference by representing the weights and activations with low-precision data types like 8-bit integer (int8) instead of the usual 32-bit floating point (float32). Reducing the number of bits means the resulting model requires less memory storage, consumes less energy (in theory), and operations like matrix multiplication can be performed much faster with integer arithmetic. It also allows to run models on embedded devices, which sometimes only support integer data types. In our model, we used 6-bit integer to store our weights.

During our designing, we also found a great usage of the LUTs. The convolutional neural network will use 83% of the LUT resources. We try to store our weights on the BRAM to reduce the use of LUT. However,these operations lead to a decrease in accuracy.

3.3 Model 1: Three-Layer Neural Network

The most basic neural network we choose is a network with three dense layers; we use LHC Jet, dataset of high-pT jets from simulations of LHC proton-proton collisions as the input to this input.

3.4 Model 2: Convolutional Neural Network

For this work, we assume that the input to CNN is a grayscale image with each pixel represented by an integer (0 to 255). This image is the input feature map to the first convolution layer, which convolves the input with n × n binary kernels. To perform the convolution, each element of the convolution output (an output feature map) is calculated by multiplying a kernel with a n × n window of the input feature map and summing the resulting values. Sweeping an n × n kernel across the input feature map generates an output feature map. The convolution operation is followed by a maximum pooling operation which reduces the size of its input feature maps by choosing the largest value out of each k × k window of each input feature map and discarding the rest. The next layer, batch normalization, normalizes its input feature maps value by value. Here, the numbers are represented as fixed-point values between -1 and +1. The non-linear function layer truncates the output feature map values of the batch normalization layer into either -1 or +1 based on their sign. This process is replicated for other convolution steps with the exception that their input feature maps are the binary outputs of the previous non-linear function layer.

The CNN is pre-trained with the MNIST database on a Nvidia GTX 3060 GPU. The trained network is used during the inference stage to classify the input images of the digits into one of ten categories (0 to 9). Our CNN contains two convolution layers and one fully connected layer. Convolution is performed with a standard 16 kernels per layer. The output of the network is a ten-element array that shows the likelihood of the input image being each of the ten digits with the highest number being the predicted digit for the input image. Figure 1 shows the structure of the CNN.

3.5 Resource Utilization

The resources used for neural networks are recorded in Table 1, and model 1 refers to the first dense neural network that we used. model 2 refers to the CNN neural network, we modified the hyperparameters, including the number of layers, the dimension of each layer, the CNN filter size, and the size of each layer so that we can make full use of the pynq board and not exceed the hardware limitation.

According to our calculation, Our model 1 contains 4389 parameters, we use 6 bits to describe each parameter, for model 2 the number of parameters is 3330. The FF used for CNN model is much more higher then model 1, this is because the CNN model has more kinds of layers, including the convolutional layer, maxpooling layer and flatten layer, which needs more logic slices.

4 VOLTAGE-BASED SIDE-CHANNEL ATTACK

Our attack method is based on the [18]. In our evaluation, we will use XADC to test the change in voltage. The XADC is the basic

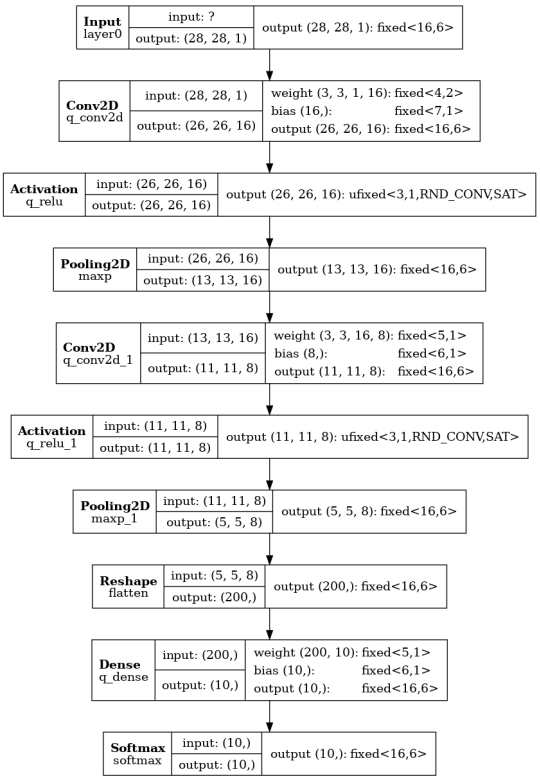

Figure 1: Model 2 structure

Table 1: Resource Utilization for Models

	BRAM_18K	DSP48E	FF	LUT	URAM
model 1	4	32	23779	42931	0
model 2	73	100	42537	40929	0
available	280	220	106400	53200	0

building block that enables analog mixed-signal (AMS) functionality. The XADC includes a dual 12-bit, 1 Mega sample per second (MSPS) ADC and on-chip sensors. It includes several on-chip sensors that support the measurement of the on-chip power supply voltages. The ADC conversion data is stored in dedicated registers called status registers. These registers are accessible through the FPGA interconnect using a 16-bit synchronous read and write port called the dynamic reconfiguration port (DRP). Figure 2 shows the structure of the neural network model kernel with an XADC kernel.

The whole design of our programme is based on the Overlay API of python, a simple Python wrapper that interfaces directly with the hardware IP blocks. We used multithreading to control different IP cores to make them work simultaneously, one thread run the XADC, and one thread run the neural network IP after the XADC begins capturing voltage drops.

The sampling rate of the XADC has been set to 1 million per second. For all of our neural network, the running time of the neural network core to deal with one piece of data is less than 0.02 seconds.

5 REVERSE ENGINEERING VIA VOLTAGE MONITORING

There is a strong relationship between the network's structure and its TDC trace based on findings in [19] and the results we found in Section 4. We can construct a reverse engineering based on this relationship. In [23] on model distillation, the author provides a simple yet powerful reverse engineering method from the model's training checkpoints, which involves retraining a model using the original model as a teacher. The student model can be a simpler and lighter version, possess the same structure as the teacher, or be a more complicated one. The process we used is listed below.

Algorithm 1 Reverse Engineering of a CNN Model

Require: Trained target CNN model M
Ensure: Reversed CNN model R
1: Inference on dataset $\mathcal{D}$ using M to get label $\mathcal{Y}'$
2: Analyze TDC trace, select structure for R
3: Train $\mathcal{R}$ on $\mathcal{D}$, $\mathcal{Y}'$; Evaluate $\mathcal{R}$
4: Compile $\mathcal{R}$ to HLS with hl4ml, generate bitstream
5: Deploy bitstream on hardware

More specifically, our reverse engineering steps are:

(1) Do inference on a dataset $\mathcal{D}$ using the target model and get labels $\mathcal{Y}'$ for $\mathcal{D}$.
(2) Based on the TDC trace collected during inference, pick a similar structure.
(3) Train in $\mathcal{D}$ and $\mathcal{Y}'$, get a reversed model $\mathcal{R}$. Evaluate the performance.
(4) Compile $\mathcal{R}$ into HLS using hl4ml toolkit and generate bitstream.
(5) Deploy the bitstream on another hardware.

In the next section, we will analyze our reverse engineering performance under different assumptions.

6 EVALUATION

This section presents the evaluation results for the victim neural networks on PYNQ-Z2.

6.1 Voltage-Based Side-Channel Attack

6.1.1 Three-Layer Neural Network. Figure 3 shows the voltage trace overview of Model 1 on PYNQ-Z2, the y-axis is the voltage on the FPGA board ranges from 1.03V to 1.01V. Similar to the paper provided before, we also found two big voltage drops in the running procedure. For the first drop, the data is sent from the host computer to the FPGA board, and the second drop shows the neural network starts running and read data at this time.

6.1.2 Two-Layer CNN. The voltage traces for the CNN are shown in Figure 4, the traces are similar to the voltage trace we found in the neural network.

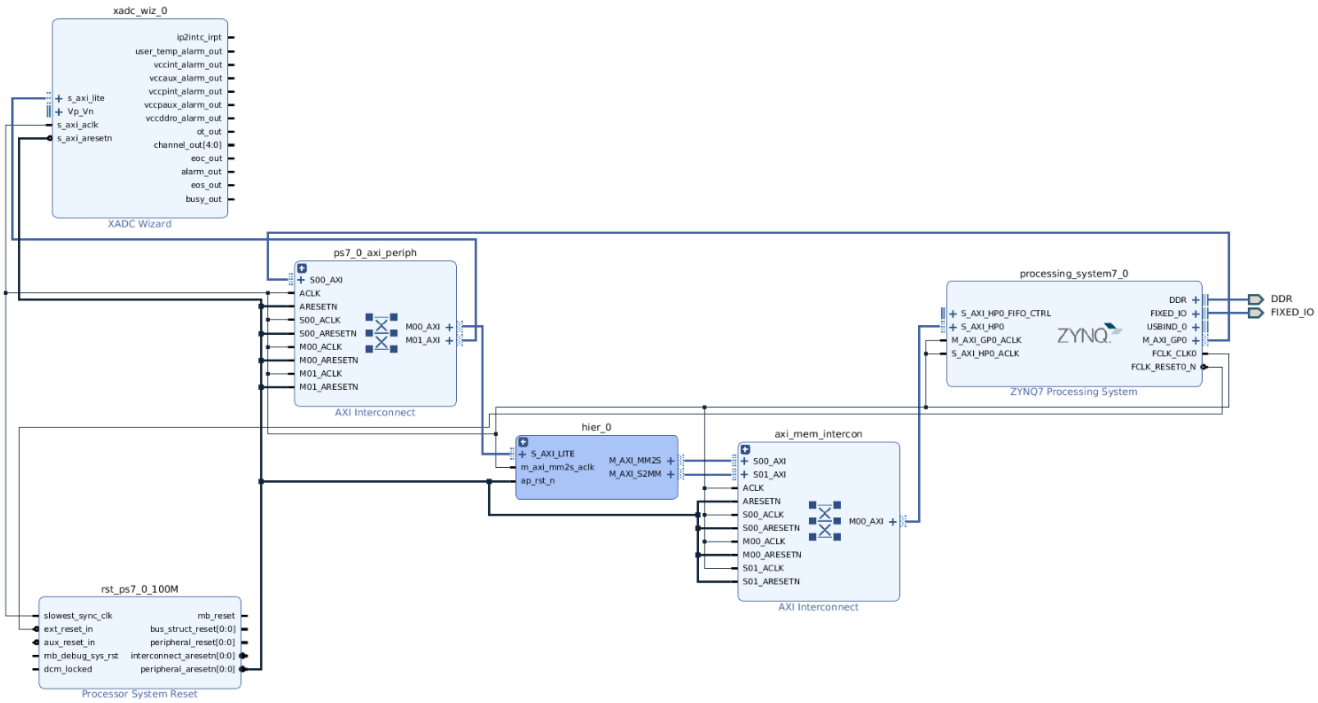

Figure 2: XADC with neural network

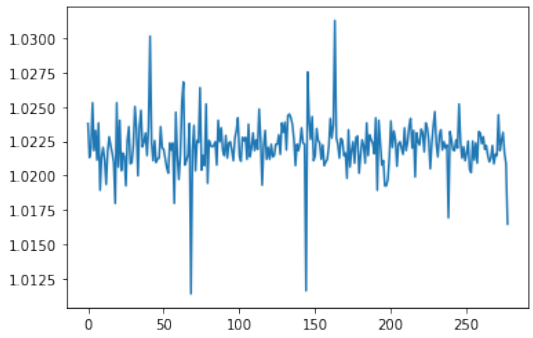

Figure 3: The voltage trace of the neural network core in PYNQ-Z2.

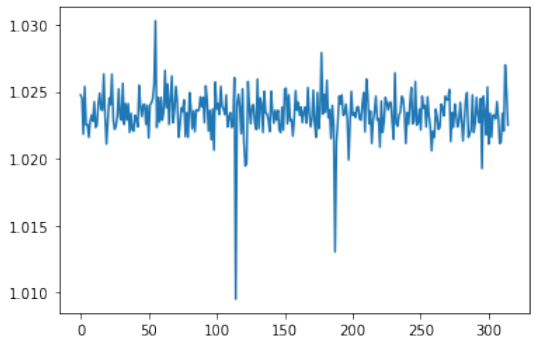

Figure 4: The voltage trace of the CNN core in PYNQ-Z2.

6.2 Auto-Triggered Remote Attacks on PYNQ-Z2

The design of the auto trigger takes into account that the two voltage trace drops, induced by the start of the host to FPGA communication or accelerator computation, are distinguishable from the idle state and other activities, as shown in Figure 3 and Figure 4.

At the beginning, the auto-trigger starts monitoring the voltage fluctuations and detects the first drop (the start of host-FPGA communication) after an unknown duration. It then will detect the second drop (the start of model computation) and then takes measurements. Then we con reconstruct the voltage drop based on the data we captured. [18] proved that different machine learning models have different voltage changes.

What we found makes this attack challenging is the duration time of computation differs greatly, we make the program run several times in completely the same circumstances. While the computation time varies from 0.017 seconds to 0.003 seconds. In this case, it is hard to decide how long the whole measurement should take. If the measurement takes too short, we may not be able to capture the whole voltage, and if it takes too long, we may not be able to store all these data.

6.3 Reverse Engineering via Voltage Monitoring

In this section, we compare the performance of three distinct model architectures in the context of reverse engineering. The models evaluated include: (i) the "Original" model, which is a standard CNN

and is depicted in Figure 1; (ii) the "Simple" model, a reduced CNN with fewer convolution and pooling layers; and (iii) the "Complex" model, which is based on Resnet-50 [3] and contains significantly more parameters than the others. Table 2 summarizes the comparative results.

The Original model, with its basic CNN structure, achieves a high accuracy of 0.9793, demonstrating its efficacy. The Simple model, with a streamlined architecture, still maintains a respectable accuracy of 0.9016. However, the Complex model, despite its advanced and parameter-heavy design and further more training time, only achieves an accuracy of 0.3050.

The comparasion of three models are further detailed by parameters, operational counts, and weight sparsity across all models. The same structure model stands out in terms of validation accuracy, which suggests our pipeline of getting the structure of the model and retrain it is workable. The similar after-training weight sparsity observed in the retrained student model, which we trained from scratch, suggests a high degree of success in reverse engineering. The attackers can leverage the retrained checkpoint to generate HLS files and also load the bitstream onto the board.

Table 2: Comparison of Different Model Structures

Structure	Original	Simple	Same	Complex
ACC	0.9793	0.9016	0.9453	0.3050
Parameter	6.4k	5.9k	6.4k	25.7M
Num of Ops	380k	102k	380k	3.8B
Weight Sparsity	0.7553	0.7639	0.7582	0.9735

7 RELATED WORK AND DISCUSSION

7.1 Defenses for Voltage-based Side-channel Attack

Facing this voltage-based side-channel attack on multi-tenant cloud FPGAs, it is difficult for the system to detect it. Thus, a feasible defense strategy is to obfuscate the power or voltage fluctuations, such as the "Active Fence" proposed by Krautter et al. [6], which places a fence between the victim and the attacker that can interfere with voltage measurements, thus increasing the number of measurements required for the attacker to successfully launch a side channel attack.

In addition, Krautter et al. [7] also proposed a bitstream checking-based defense measure, which analyzes and checks the bitstream files to be loaded into the FPGA, evaluating their potential for malicious run-time behavior. However, these methods all incur additional high costs; therefore, a low-cost defense method still needs further exploration.

7.2 Defenses for Reverse Engineering

In our comparison of the three models, we noted that while the model with the same structure achieves the highest accuracy, the simpler model still displays competitive performance. This observation is consistent with findings from [23], which demonstrate that a student model can surpass the teacher model in performance, despite having a different structure. The extensive practice of model distillation since [4] has equipped modern attackers with the tools necessary for reverse engineering by analyzing both the output and intermediate vectors between layers.

In fact, attackers do not necessarily require the exact structure of the original model to reverse engineer a simpler yet effective version. In the field of large language models, this form of distillation is already being applied [21], where more powerful models like GPT-4 serve as teachers or judges. To safeguard the intellectual property of such models, strategies like embedding a watermark [22], implementing input preprocessing, and conducting adversarial training [10] are becoming increasingly important.

7.3 Poor Performance of Larger Model

The Complex model we use in section 5 only achieves an accuracy of 0.3050, despite its advanced design and its training accuracy is 0.88. This suggests that Resnet-50 may be overfitting or inefficient for this specific reverse engineering task on MNIST dataset. Originally designed for processing larger RGB images with a resolution of 224x224[3], Resnet-50 is being applied to significantly smaller, grayscale images of 28x28 in our task, which likely contributes to its suboptimal performance.

Additionally, the model's extensive parameter count is excessive for a relatively simple dataset like MNIST. This is consistent with findings from [14], where the CLIP model excels in various vision tasks but underperforms on the MNIST dataset. This underperformance underscores that MNIST, with its unique distribution distinct from typical vision datasets, poses specific challenges that are not effectively addressed by larger models. Also, in terms of distillation, people are often using a smaller or same model as a student model[23], it may not be wise to use a more complicated model.

8 CONCLUSION

This work explores the security threats faced by machine learning models in the cloud FPGA multi-tenant environment and proposes an attack method that combines voltage-based side-channel attacks and voltage monitoring reverse engineering. By monitoring the voltage fluctuations of the FPGA chip, attackers can infer the structure and parameters of the machine learning models deployed on the FPGA, thereby achieving the theft and cloning of these models.

Our research results demonstrate that malicious attackers in the cloud FPGA multi-tenant environment can still obtain critical information through voltage side channels and steal or replicate machine learning models deployed on FPGAs, even without direct access to the victim's design. In addition to discussing the aforementioned attack method, we also provide references to defend against such attacks both in the TDC tracing stage and in the reverse engineering stage and optimize the deployment of machine learning models on FPGAs to conserve more hardware resources.

ACKNOWLEDGMENTS

This work was supported in part by Fu Foundation School of Engineering and Applied Science.

REFERENCES

[1] FastML Team. 2023. *fastmachinelearning/hls4ml.* https://doi.org/10.5281/zenodo.

1201549

[2] Kunihiko Fukushima. 1980. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics* 36 (1980), 193–202. https://doi.org/10.1007/BF00344251

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. *Deep Residual Learning for Image Recognition.* Technical Report. https://doi.org/10.48550/arXiv.1512.03385 arXiv:1512.03385 [cs.CV].

[4] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML]

[5] J. J. Hopfield. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences* 79, 8 (April 1982), 2554–2558. https://doi.org/10.1073/pnas.79.8.2554

[6] J. Krautter, D. R. E. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori. 2019. Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* Westminster, CO, USA, 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942094

[7] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. 2019. Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud. *ACM Trans. Reconfigurable Technol. Syst.* 12, 3, Article 12 (aug 2019), 26 pages. https://doi.org/10.1145/3328222

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, Vol. 86. 2278–2324. https://doi.org/10.1109/5.726791

[10] Gabriel Resende Machado, Eugênio Silva, and Ronaldo Ribeiro Goldschmidt. 2021. Adversarial Machine Learning in Image Classification: A Survey Toward the Defender's Perspective. *Comput. Surveys* 55, 1 (Nov. 2021), 1–38. https://doi.org/10.1145/3485133

[11] J. M. Mbongue, S. K. Saha, and C. Bobda. 2021. Performance Study of Multi-tenant Cloud FPGAs. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* Portland, OR, USA, 168–171. https://doi.org/10.1109/IPDPSW52791.2021.00032

[12] Warren S. McCulloch and Walter Pitts. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* 5 (1943), 115–133. https://doi.org/10.1007/BF02478259

[13] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier. 2021. Remote Power Side-Channel Attacks on BNN Accelerators in FPGAs. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE).* Grenoble, France, 1639–1644. https://doi.org/10.23919/DATE51398.2021.9473915

[14] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV]

[15] Frank Rosenblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65, 6 (1958), 386–408. https://doi.org/10.1037/h0042519

[16] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Representations by Back-Propagating Errors. *Nature* 323 (1986), 533–536. https://doi.org/10.1038/323533a0

[17] S. Tian, S. Moini, D. Holcomb, R. Tessier, and J. Szefer. 2023. A Practical Remote Power Attack on Machine Learning Accelerators in Cloud FPGAs. In *2023 Design, Automation Test in Europe Conference Exhibition (DATE).* Antwerp, Belgium, 1–6. https://doi.org/10.23919/DATE56975.2023.10136956

[18] Shanquan Tian, Shayan Moini, Daniel Holcomb, Russell Tessier, and Jakub Szefer. 2023. A Practical Remote Power Attack on Machine Learning Accelerators in Cloud FPGAs. In *2023 Design, Automation Test in Europe Conference Exhibition (DATE).* 1–6. https://doi.org/10.23919/DATE56975.2023.10136956

[19] Shanquan Tian, Shayan Moini, Adam Wolnikowski, Daniel Holcomb, Russell Tessier, and Jakub Szefer. 2021. Remote Power Attacks on the Versatile Tensor Accelerator in Multi-Tenant FPGAs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* 242–246. https://doi.org/10.1109/FCCM51124.2021.00037

[20] Shanquan Tian and Jakub Szefer. 2019. Temporal Thermal Covert Channels in Cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19).* Association for Computing Machinery, New York, NY, USA, 298–303. https://doi.org/10.1145/3289602.3293920

[21] Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023. Zephyr: Direct Distillation of LM Alignment. arXiv:2310.16944 [cs.LG]

[22] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (ICMR '17).* ACM. https://doi.org/10.1145/3078971.3078974

[23] Chaofei Wang, Qisen Yang, Rui Huang, Shiji Song, and Gao Huang. 2022. Efficient Knowledge Distillation from Model Checkpoints. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 607–619. https://proceedings.neurips.cc/paper_files/paper/2022/file/03e0712bf85ebe7cec4f1a7fc53216c9-Paper-Conference.pdf

[24] Jun Wang and B. Malakooti. 1992. A feedforward neural network for multiple criteria decision making. *Computers Operations Research* 19, 2 (1992), 151–167. https://doi.org/10.1016/0305-0548(92)90089-N