2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST) | 979-8-3503-4099-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/AsianHOST59942.2023.10409396

NNLeak: An AI-Oriented DNN Model Extraction Attack through Multi-Stage Side Channel Analysis

Ya Gao*, Haocheng Ma*, Mingkai Yan*, Jiaji He*§, Yiqiang Zhao*, and Yier Jin†
*School of Microelectronics, Tianjin University
†University of Science and Technology of China
{gaoyaya, hc_ma, youngminl2021, dochejj, yq_zhao}@tju.edu.cn, yier.jin@ustc.edu.cn

***Abstract*—Side channel analysis (SCA) attacks have become emerging threats to AI algorithms and deep neural network (DNN) models. However, most existing SCA attacks focus on extracting models deployed on embedded devices, such as microcontrollers. Accurate SCA attacks on extracting DNN models deployed on AI accelerators are largely missing, leaving researchers with an (improper) assumption that DNNs on AI accelerators may be immune to SCA attacks due to their complexity. In this paper, we propose a novel method, namely NNLeak to extract complete DNN models on FPGA-based AI accelerators. To achieve this goal, NNLeak first exploits simple power analysis (SPA) to identify model architecture. Then a multi-stage correlation power analysis (CPA) is designed to recover model weights accurately. Finally, NNLeak determines the activation functions of DNN models through an AI-oriented classifier. The efficacy of NNLeak is validated on FPGA implementations of two DNN models, including multilayer perceptron (MLP) and LeNet. Experimental results show that NNLeak can successfully extract complete DNN models within 2000 power traces.**

***Index Terms*—Deep Neural Network, Side Channel Analysis, DNN Model Extraction**

I. Introduction

Deep neural networks (DNNs) have become the dominant force in artificial intelligence (AI) due to their robust feature extracting and learning capabilities. Numerous DNN models are deployed on GPUs and other AI accelerators, which achieve great success in real-world applications, such as medical diagnosis, autonomous vehicles and robotics. Typically, the training process of a DNN is expensive and requires a large database, which makes DNN's architecture and parameters not only intellectual properties but also important security assets. As a consequence, a well-trained DNN model with huge commercial value is an attractive target for attackers. Nowadays, model extraction through SCA emerges as a serious threat to edge devices powered by AI algorithms. In this process, an adversary carries out model extraction by monitoring side channel leakage, such as timing, power consumption and electromagnetic (EM) emanations [1], [2], [3].

Model extraction through SCA attacks can be divided into two categories, including alternative construction and exact extraction. In alternative construction, the attacker aims to obtain an alternative DNN, which has similar accuracy with the victim DNN. While in exact extraction, the recovered model has the same architecture and parameters as the victim DNN. Here, we summarize existing exact extraction through SCA attacks, as listed in Table I. Batina et al. combine SPA and CPA techniques to recover the architecture and parameters of MLPs and a seven-layer convolutional neural network (CNN) [4]. For 32-bit floating point parameters in DNNs, they consider the attack successful if the recovered parameter is correct up to two decimal places. While this may cause insufficient precision of the data and reduce the effectiveness of model extraction. Then Takatoi et al. extend this work to identify activation functions (AFs) implemented with constant time [5]. Note that these two attacks focus on DNN models deployed on the microcontrollers. Recently, Yoshida et al. exploit CPA to extract DNN models deployed on FPGA platforms. Due to the binary shifted problems, they only recover partial weights of a small-scale MLP with 9 neurons and two systolic arrays [6], [7]. Based on differential power analysis (DPA) techniques, Dubey et al. recover the 1-bit parameters from the hardware of the adder tree, which stores multiplication results in the binarized neural network (BNN) [8].

From the above analysis, we observe that existing techniques of model extraction confront following challenges. Most of them are applicable to DNN models running on simple software platforms. While for those hardware implementations, many techniques can only recover small-scale DNNs or partial modules of AI accelerators. In addition, the recovery weights of DNNs are limited by the binary shifted problem, resulting in false positive results. These drawbacks make existing techniques less practical in real-world applications.

To this end, we propose an extraction method called NNLeak (see Figure 1). NNLeak adopts a multi-stage side channel attack strategy, which combines both SPA and CPA, to identify the architecture and recover weights of the DNN models. Also, NNLeak incorporates the idea of AI attacking AI to recover complete DNN models on FPGA-based AI accelerators. To demonstrate their effectiveness, we deploy two DNN models, MLP [9] and LeNet [10], which are widely used in the field of computer vision, on an FPGA platform. Experimental results prove that NNLeak is able to recover not only the AI model architecture (e.g., the number of layers and neurons) but also all parameters (e.g., weights and AFs) precisely.

The main contributions of this paper are listed as follows.

§Corresponding author.

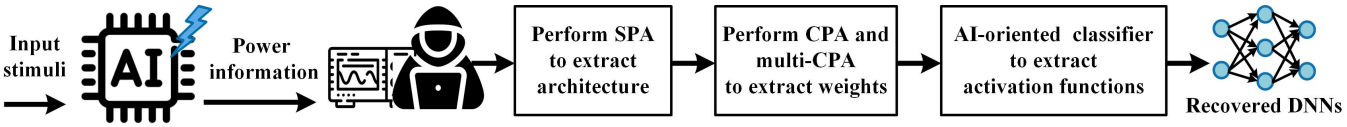


Fig. 1. The proposed NNLeak.

TABLE I
EXISTING SCA BASED MODEL EXACT EXTRACTION ATTACKS

Work	Physical Target	Model	Attack Target		
			architecture	weights	AF
Batina. [4]	microcontroller	MLP (50-30-20-50), CNN(7-layer)	✓	✓	✓
Yoshida [6]	FPGA	MLP (2-5-2)	×	✓	×
Yoshida [7]	FPGA	Systolic array*	×	✓	×
Dubey [8]	FPGA	Adder tree*	×	✓	×
Takatoi [5]	microcontroller	MLP (3-3-9), MLP (3-9-9)	×	×	✓
our work	FPGA	MLP (62-20-10), LeNet(7-layer)	✓	✓	✓

* Data has not been reported.

- A practical framework, namely NNLeak, is proposed for exact extraction of DNN models. Its effectiveness is validated using two real-world DNNs implemented on FPGA platforms.
- One multi-stage SCA method is proposed. First, SPA is utilized for architecture identification. Then multi-stage CPA is used to solve the common binary-left or binary-right shifted problem in CPA attacks on weight multiplication, and thus extract the weight precisely.
- Inspired by the idea of AI attacking AI, an AI-oriented AF classifier is proposed for constant time AF recognition.
- Experimental results show that, we can precisely recover DNN models, including the number of layers, neurons, weights and the type of AFs, from an FPGA-based AI accelerator.

The rest of the paper is organized as follows. Section II briefly introduces some background on DNNs and SCAs. Section III describes our NNLeak of the model extraction attack. Section IV and Section V present the experimental results of the proposed attack. Section VI concludes this paper finally.

II. BACKGROUND

A. Deep Neural Network

Deep neural networks (DNNs) are computational models that include different types and numbers of layers, as well as parameters such as weights, biases and activation functions. Neurons are important blocks in DNNs for extracting data features and Equation (1) depicts their specific operations.

$$h_k = f\left(\sum_{i=0}^{n-1}(x_i \cdot W_{xihk}) + b_k\right) \tag{1}$$

where x_i is the output value of the previous layer, W_{xihk} is the weight corresponding to x_i, and b_k is the bias. $f(x)$ is the nonlinear activation function, which is the last operation of the neuron and is used to extract further feature information.

MLPs are the most basic DNNs. A typical MLP consists of three fully connected layers named input layer, hidden layer, and output layer. As the scale of DNNs' application scenarios grows, the development of MLPs is limited and more complex CNNs are proposed and widely used. CNNs are made up of convolutional layers, pooling layers, and fully connected layers that capable of extracting more intricate and abstract high-level features from inputs.

B. Side Channel Analysis

SCA attacks aim to recover sensitive information by monitoring power consumption, EM emanations, timing, and other information inadvertently released by devices. Early SCA attacks are widely used to steal keys in cryptographic algorithms [11]. Typically, there are two main types of common SCAs, including simple power analysis (SPA) and differential power analysis (DPA).

SPA performs coarse-grained analysis of a single or few traces with the help of visual inspection, expecting to obtain sensitive information. In DPA, the adversary will first build the leakage model to map different hypotheses of the secret after collecting a set of traces. The traces are then compared with the leakage model through statistical methods to determine the correct hypothesis. CPA is the evolved version of DPA, it regards Pearson correlation as the distinguisher to reveal the correct hypothesis, as given in Equation (2).

$$r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i} - \overline{h}_i) \cdot (t_{d,j} - \overline{t}_j)}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \overline{h}_i)^2 \cdot (t_{d,j} - \overline{t}_j)^2}} \tag{2}$$

Assuming that the attacked weight is 8-bit, the attacker has a total of 2^8 weight hypotheses, i.e., i = 2^8. D represents the number of power measurements. $h_{d,i}$ denotes the theoretical leakage model after the input d is calculated with the hypothesis i. Usually we use the Hamming distance (HD) or Hamming weight (HW) model. $t_{d,j}$ is the power consumption value of the d-th power measurement at the j-th time point. The calculation result $r_{i,j}$ count the correlation value of all weight hypotheses with the real power measurements during the attack time period, where the correct weight hypothesis will stand out with a higher value.

III. Model Extraction Techniques

In this section, we will introduce the overall framework of NNLeak. Based on a series of techniques, it recovers the architecture, weights and activation functions in turn.

A. Threat Model

In this work, we assume that DNN models are deployed on FPGAs for hardware acceleration, which helps execute the inference process. Although the attacker does not know the training dataset, he has physical access to FPGAs. By importing carefully designed input stimuli, the attacker collects power consumption traces to extract architecture and parameters of DNN models. We further assume that DNN models do not equip with side channel protection mechanisms [12].

B. Architecture Recovery

To perform exact extraction, the first step is to identify the DNN architecture, e.g., the number of layers and neurons. Inspired by previous work [13], NNLeak applies SPA to realize the objective of architecture recovery. During the inference procedure, involving operations will execute layer by layer and process different amounts of data. This results in different characteristics of power consumption that are related to model architecture, making clear boundaries in the power traces. For each layer, neurons are also executed one by one. All neurons perform the same operation, so the power consumption traces have a clear regularity that can be further localized to the execution window of each neuron.

C. Weights Recovery

After extracting the DNN architecture, NNLeak tries to extract model weights layer by layer. In this step, the attacker targets the arithmetic multiplication of a known input x_i with a secret weight W_{xihk}. The multiplication result P_{xihk} is stored in registers and causes bit transitions, resulting in data-dependent power leakages. Figure 2 shows the calculation of the first neuron in the hidden layer, it contains i weights associated with i input layer neurons. For simplicity, we take the weight W_{x1h1} as an instance to describe weights extraction. Given random input stimuli, multiple sets of power traces are collected from the hardware implementation of the DNN model. The leakage model is built based on the HD of the product P_{x1h1}. Note that P_{x1h1} is the product of weight W_{x1h1} and received input x_1. According to Equation (2), we calculate the Pearson correlation between the leakage model and collected traces. The true value of W_{x1h1} often has a high correlation and can be separated from other false hypotheses.

If false hypotheses have 2^n-fold relationship with the correct weight, binary representations of their products will have a shifted relationship. In this scenario, the amount of 1 in binary terms keeps the same while their positions change by moving left or right. Equation (3) lists the products for the same input 126 with several hypotheses of the target weight, e.g., 16, 32 and 64. Here we regard 32 as the correct weight. Considering registers switch from reset states to product results, the above three weights point to the same leakage model. As a result,

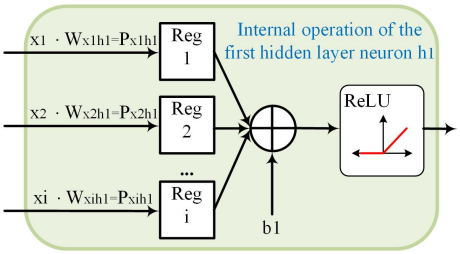


Fig. 2. Internal operation of the first hidden layer neuron h_1.

no clear distinction can be seen from their correlation traces. The weights recovered by CPA may be false positive results, i.e., binary left-shifted or binary right-shifted values of the true weights.

$$\begin{aligned} 126\times 16(00010000)_2 &= 1026 = (0000011111100000)_2 \\ 126\times 32(00100000)_2 &= 4032 = (0000111111000000)_2 \\ 126\times 64(01000000)_2 &= 8064 = (0001111110000000)_2 \end{aligned} \quad (3)$$

We propose a multi-stage CPA method to separate the true weight from false positive results. Specifically, we truncate higher bits of the obtained product in turn to build leakage models. They reflect power variations caused by data bits of partial products. As shown in Equation (3), when the higher 4-bit is truncated, leakage models of false hypothesis 64 start to diverge from other guessed weights. When lower 11-bit retains after truncation operations, leakage models related to correct weight 32 can be distinguished from false guess 16. For false hypotheses, the correlation coefficients between their leakage models and power traces will gradually decrease. Correspondingly, the true value of weight will stand out in this procedure. Therefore, multi-stage CPA can eliminate the effect of the shifted problem. We take the multiplication result of this true weight as the current state of registers. The multi-stage CPA will execute again to extract the next target.

All the weights corresponding to each neuron are extracted by the CPA described above. Since DNNs are executed layer by layer, the attacker is allowed to use the weights and AF recovered from the previous layer with the input stimuli to compute the inputs for the next layer. Further, the inputs can be used to construct a new leakage model to extract the weights of the next layer.NNLeak iterates the process until all weights are extracted.

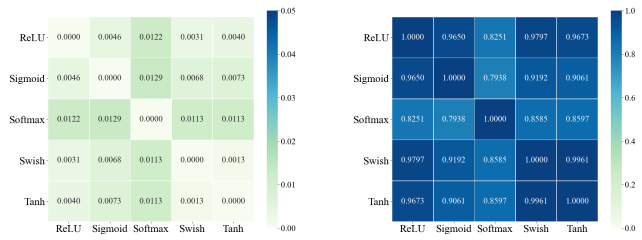
D. Activation Function Recovery

The common AFs in DNNs are ReLU, Sigmoid, Tanh, Softmax, Swish and so on. Their identification is essentially equivalent to a classification problem, which is also an important step in exact extraction. Previous works like timing SCA [4] and direct observation [5] are effective in targeting AI models on software-programmed microcontrollers. However, their efficacy of model extraction targeting FPGA-based AI accelerators is limited. FPGAs often use piecewise linear approximation (PWL) and lookup tables to implement AFs. These implementations replace various and intricate nonlinear functions with simple linear functions or straightforward data mappings. Thus, the power traces of AFs will become

similar and not easily identified by direct observation. Also, in previous methods, attackers modulate the input stimuli of AFs to amplify their power variations. This is difficult to achieve stimuli modulation in DNN models with large-scale parameters.

We implemented five typical activation functions executed in constant time, including ReLU, Sigmoid, Tanh, Softmax, and Swish [14], [15], on a 16-bit quantized MLP model using the PWL method. We then deploy the model on an FPGA. In order to compare their power variations, we select the Euclidean distance and cosine similarity as evaluation criteria, as shown in Figure 3. The Softmax function has a different power distribution since it executes more arithmetic operations. The remaining functions are not distinguished easily due to similar power distributions.

Inspired by the idea of AI attacking AI, NNLeak exploits AI-oriented techniques to extract the AFs on FPGA-based AI accelerators. Various classifiers like MLP, support vector machines (SVM), Naive Bayes, and CNN satisfy our requirements. For proof of concept, here we choose MLP to solve the classification problem. For each AF, we collect 500 power traces as a database to train our classifier. Our classifier is able to identify the AF of the DNN with an accuracy of 99.6%. In the practical attack, for any unknown DNNs, the attacker can implement several common AFs based on the quantization bits of DNN's parameters and train a classifier. By extracting the traces from the AF execution window and sending them to the classifier, the type of AFs is determined.



(a) Euclidean distance (b) cosine similarity

Fig. 3. The degree of difference between AFs.

IV. Case Study: MLP Model Extraction

A. Experimental Setup

To evaluate the proposed NNLeak, we implement two exemplary DNNs, an MLP and a LeNet, on a Sakura-G platform (see Figure 4). These designs are operated with a clock frequency of 50MHz and a supply voltage of 1.8V. The power consumption is measured using an on-board amplifier. The collected signals are sampled at 2.5 GS/s by an oscilloscope with a 500 MHz bandwidth. To eliminate the measurement noise, we take the average of 16 power traces as the data for analysis.

B. Hardware Implementation of MLP

We choose an MLP for MNIST purpose. This MLP has 3 layers with 62, 20 and 10 neurons, respectively. All the

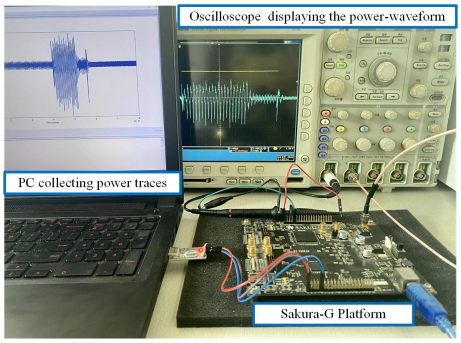


Fig. 4. The experimental setup for NNLeak.

parameters are expressed by 16-bit fixed point, and the high 8-bit is set to either 11111111 or 00000000. The MLP gets its inputs from the MNIST database, which contains 60000 training images and 10000 testing images. Using feature conditioning algorithms, each 28×28 pixel image is reduced into 62 features.

Table II describes the execution process of MLP. Starting from the first neuron in the hidden layer, all neurons perform the same multiplication, accumulation and activation function operations one after another. Therefore, the power traces of MLP should theoretically have obvious regularity and boundaries. The number of layers and neurons can be further confirmed.

TABLE II
THE EXECUTION PROCESS OF MLP

operation	cycles
$x_1 \times W_{x1h1}$, $x_2 \times W_{x2h1}$.....$x_{62} \times W_{x62h1}$	1
Accumulation	1
Activation Function	1
.....	...
$x_1 \times W_{x1h20}$, $x_2 \times W_{x2h20}$.....$x_{62} \times W_{x62h20}$	1
Accumulation	1
Activation Function	1

C. Model Extraction on MLP

Figure 5 shows the complete power traces of the MLP. Based on the magnitude of the power peak, NNLeak can distinguish the hidden layer and the output layer. After inputs are sent to the MLP, neurons in the hidden layer start the computation sequentially, showing 20 similar power peaks, which means 20 neurons. The results of the hidden layer will be sent to the output layer for further computation, and the 10 similar power peaks represent 10 neurons in the output layer. As a result, NNLeak successfully extracts the number of layers and neurons in MLP through SPA.

In the step of extracting weights, the HD model will be built first based on hypotheses of the weight. As the MLP is 16-bit quantized, there are 2^{16} hypotheses for one weight in theory. Since the high 8 bits of the parameters are fixed as two cases, the candidate hypotheses are compressed to 2×2^8. Next, we send 2000 random inputs to the MLP and collect power traces.

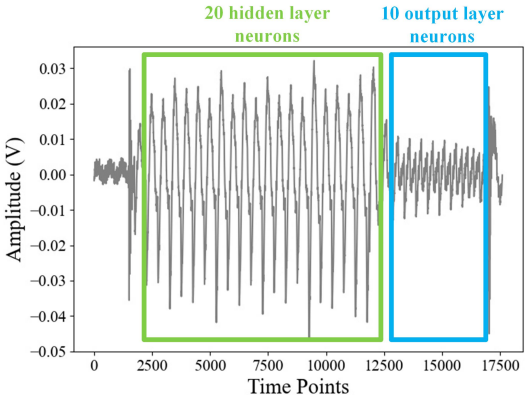
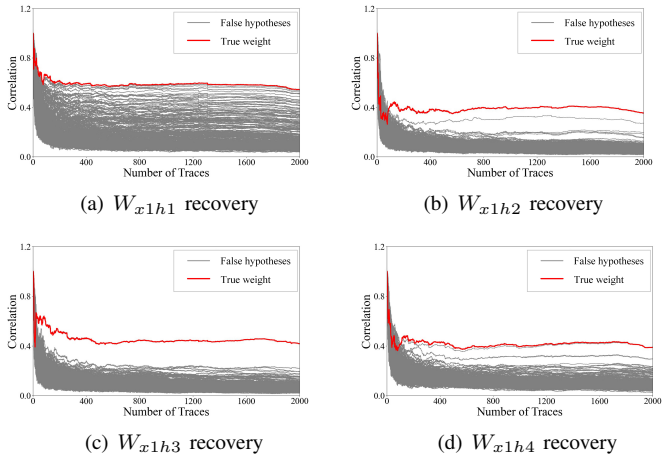


Fig. 5. Power traces of the MLP execution process.

Finally, we locate the multiplication operation of each neuron by means of windowing and perform CPA.

According to the execution characteristics of the MLP, given a set of inputs, all the 20 weights multiplied with x_1 can be extracted with 100% accuracy directly. The CPA results for the weights associated with the input x_1 are shown in Figure 6, with the x-axis representing the number of traces required for a successful attack and the y-axis representing the correlation coefficient. We demonstrate the attack results for four of the weights. W_{x1h1} to W_{x1h4} represent the weights between the hidden layer neurons h_1 to h_4 and x_1, respectively. The red line represents the true weight and the other gray lines represent false hypotheses. It can be seen that only after about 100 traces, the correlation of the true weight outweighs false hypotheses in a conclusive manner. Iterating the same attack for each multiplication, other weights can also be extracted exactly.



(a) W_{x1h1} recovery (b) W_{x1h2} recovery

(c) W_{x1h3} recovery (d) W_{x1h4} recovery

Fig. 6. CPA results on the weights of MLP.

V. Case Study: LeNet Model Extraction

A. Hardware Implementation of LeNet

We then extend the attack's applicability from MLPs to CNNs. Although LeNet is relatively simpler than modern CNN models, it is indeed the first network used for the commercial purpose, such as handwritten digit recognition on checks. The LeNet we selected consists of three convolutional layers, two max pooling layers, and two fully-connected layers. It also takes the MNIST database as the input. All images are padded with zero to 32×32 pixels.

We choose a hardware code of LeNet [10], where all the parameters are 8-bit quantized. Due to the huge amount of parameters in LeNet, the code utilizes a pipeline structure to accommodate layers with different parallelism. Specifically, the multiplication operations of a convolution kernel are typically performed in parallel, whereas the different convolutional, pooling and fully connected layers are performed sequentially using a pipelined structure. This method can improve the overall performance and increase the throughput of LeNet.

Input data and weights are stored in row buffers and on-chip parameter buffers, respectively. The convolution operation is performed by processing elements (PEs), which are the basic units for convolutional multiplication and consist of a set of Multiply and Accumulates (MACs). During the PE execution (see Figure 7), the weights are preloaded, followed by reading input data using row buffers and performing matrix multiplication.

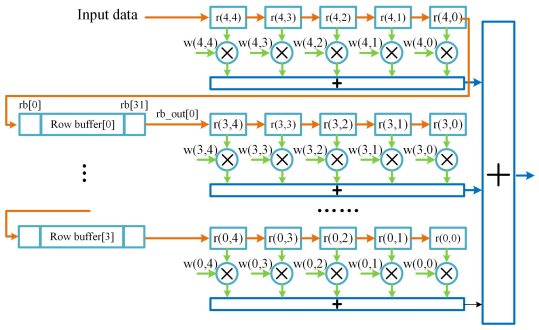


Fig. 7. The PE execution.

B. Model Extraction on LeNet

The experimental setup for model extraction on LeNet is the same as for MLP. Figure 8 depicts the execution process after a portion of input data is loaded into the LeNet model. The 1024 input data are sequentially sent to row buffers and start performing convolution, activation function and pooling operations. These operations can be identified in the power traces. The results of the pooling layer are then provided to the next convolutional layer.

Next, NNLeak executes the weight extraction attack from the weight of the first convolutional layer. The input data are sequentially multiplied with the weight of the convolution kernel in turn. The 16-bit product of LeNet is stored in registers. We also provide 2000 input data and build an HD model of registers based on 2^8 hypotheses of the weight. Then, we perform CPA on power traces and obtain the attack results for the first weight associated with the first input data, as shown in Figure 9(a). However, the initial CPA result is not robust, with high correlation positions clustering several candidate hypotheses. Experiment result shows the true weight 8'd4 (the red line) is hidden, which means the result faces the problem of binary-left or binary-right shifted.

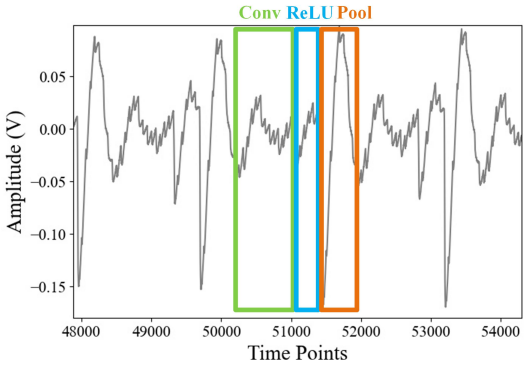
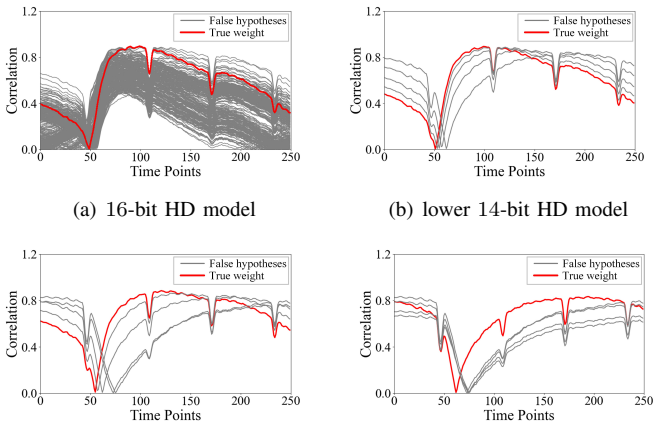


Fig. 8. The execution process of LeNet.

To recover the true weight, we further adopt the multi-stage CPA proposed in Section III. Five hypotheses (8'd4, 8'd8, 8'd16, 8'd32, 8'd64) with the highest correlation are selected for further finding the true value. We build HD models and carry out CPA attacks by sequentially truncating from the highest bit according to the products of these five hypotheses. Figure 9 shows the results of each stage in the multi-stage CPA attack. It can be clearly seen that 8'd4 starts to highlight from the CPA result of the lower 12-bit HD model. Continuing the truncation to lower 10-bit, the correlation of the HD model of 8'd4 is much higher than the other false positive results. At this point, we solve the binary-left or binary-right shifted problem and extract the correct weights 8'd4 precisely.



(a) 16-bit HD model (b) lower 14-bit HD model

(c) lower 12-bit HD model (d) lower 10-bit HD model

Fig. 9. CPA results on the weight of LeNet.

As for the pooling layer, we use the maximum pooling layer by default for most CNNs deployed on the FPGA, which requires only logical value comparison and is well-suited for FPGA computing. Finally, the fully connected layer is recovered using the same method as the MLP.

VI. Discussion and Conclusion

As more DNNs are deployed on AI accelerators for Internet of Things (IoT) and edge devices, the security and privacy of AI models should be considered. In this paper, the proposed NNLeak precisely extracts the architectures and parameters of two sample DNN models deployed on an FPGA platform using an AI-oriented multi-stage SCA approach. NNLeak demonstrates the urgency for DNN model protection on AI accelerators. However, as the model's scale grows, the time cost of exact extraction will increase significantly, as will the threshold of traces required to obtain an acceptable success rate. In the future, we will adopt exact extraction to assist alternative construction, extending the attack to DNN models in real-world devices such as smart home security systems, facial recognition security cameras, and other devices. Meanwhile, research on the defense mechanism of AI accelerators needs to be enhanced to improve the security of DNNs.

Acknowledgements

This work is supported in part by the National Key R&D Program of China (Grant No. 2021YFB3100903).

References

[1] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Poster: Recovering the input of neural networks via single shot side-channel attacks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2657–2659.

[2] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "Deepem: Deep neural networks model recovery through em side-channel information leakage," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 209–218.

[3] Y.-S. Won, S. Chatterjee, D. Jap, A. Basu, and S. Bhasin, "Wac: First results on practical side-channel attacks on commercial machine learning accelerator," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, 2021, pp. 111–114.

[4] L. Batina, S. Bhasin, D. Jap, and S. Picek, "{CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 515–532.

[5] G. Takatoi, T. Sugawara, K. Sakiyama, Y. Hara-Azumi, and Y. Li, "The limits of sema on distinguishing similar activation functions of embedded deep neural networks," *Applied Sciences*, vol. 12, no. 9, p. 4135, 2022.

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

[7] K. Yoshida, M. Shiozaki, S. Okura, T. Kubota, and T. Fujino, "Model reverse-engineering attack against systolic-array-based dnn accelerator using correlation power analysis," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 104, no. 1, pp. 152–161, 2021.

[8] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 197–208.

[9] https://gitlab.com/hadi_sfr/verilog_neural_network.

[10] https://github.com/djtfoo/lenet5-verilog.

[11] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[12] S. Maji, U. Banerjee, S. H. Fuller, and A. P. Chandrakasan, "A threshold-impiementation-based neural-network accelerator securing model parameters and inputs against power side-channel attacks," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 518–520.

[13] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Sca strikes back: Reverse engineering neural network architectures using side channels," *IEEE Design & Test*, 2021.

[14] I. Kouretas and V. Paliouras, "Hardware implementation of a softmax-like function for deep learning," *Technologies*, vol. 8, no. 3, p. 46, 2020.

[15] H. Amin, K. M. Curtis, and B. R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," *IEE Proceedings-Circuits, Devices and Systems*, vol. 144, no. 6, pp. 313–317, 1997.