NNReArch: A Tensor Program Scheduling Framework Against Neural Network Architecture Reverse Engineering

Yukui Luo, Shijin Duan, Cheng Gongye, Yunsi Fei, and Xiaolin Xu
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA, USA

Abstract—Architecture reverse engineering has become an emerging attack against deep neural network (DNN) implementations. Several prior works have utilized side-channel leakage to recover the model architecture while the target is executing on a hardware acceleration platform. In this work, we target an open-source deep-learning accelerator, Versatile Tensor Accelerator (VTA), and utilize electromagnetic (EM) side-channel leakage to comprehensively learn the association between DNN architecture configurations and EM emanations. We also consider the holistic system – including the low-level tensor program code of the VTA accelerator on a Xilinx FPGA, and explore the effect of such low-level configurations on the EM leakage. Our study demonstrates that both the optimization and configuration of tensor programs will affect the EM side-channel leakage.

Gaining knowledge of the association between the low-level tensor program and the EM emanations, we propose NNReArch, a lightweight tensor program scheduling framework against side-channel-based DNN model architecture reverse engineering. Specifically, NNReArch targets reshaping the EM traces of different DNN operators, through scheduling the tensor program execution of the DNN model so as to confuse the adversary. NNReArch is a comprehensive protection framework supporting two modes, a *balanced mode* that strikes a balance between the DNN model confidentiality and execution performance, and a *secure mode* where the most secure setting is chosen. We implement and evaluate the proposed framework on the open-source VTA with state-of-the-art DNN architectures. The experimental results demonstrate that NNReArch can efficiently enhance the model architecture security with a small performance overhead. In addition, the proposed obfuscation technique makes reverse engineering of the DNN architecture significantly harder.

I. INTRODUCTION

Neural network (NN) has found important use in different application domains, such as object detection, big data analytic, and semantic recognition. To improve the inference capability of NN models, deep neural network (DNN) is proposed, which employs larger model size for tackling more complicated tasks. Although DNNs are demonstrated as promising for different tasks, their large model sizes become a bottleneck for performance and applicability. To mitigate these issues, different methods have been proposed to accelerate the execution of DNN models [1–5]. For example, modern FPGAs have been widely deployed to provide acceleration for DNNs on both edge devices and cloud infrastructures.

A DNN model can be represented as a directed acyclic graph (DAG) composed of operation nodes and connections, thus its implementation can be mapped accordingly to FPGA hardware components (e.g., look-up-table and DSP) with FPGA-DNN development tools. The most commonly used frameworks are the Xilinx deep learning processor unit (DPU) [2], and the open-source versatile tensor accelerator (VTA) [6], which can provide end-to-end optimization for FPGA-DNN implementation. Although such FPGA-based DNN acceleration framework provide significant performance improvement, they also create a new attack surface, where an adversary can either manipulate the inference of DNN models [7, 8], or illegally extract (i.e., using side-channel analysis) the critical parameters of a DNN model, such as its architecture [9, 10]. Since the construction of high-performance DNN models involves expensive data collection and training procedures, thus their model parameters like the architecture should be highly protected.

This paper studies the vulnerability of DNN model architecture against side-channel-based DNN model extraction attack on FPGA accelerators. Specifically, we explore the association between electromagnetic (EM) side-channel leakage of DNN model implementation on FPGAs. To draw generic conclusions, we use state-of-the-art open-source FPGA acceleration framework, VTA [11], and explore the internal causes of DNN model architecture-relevant side-channel leakage from the low-level program code. Correspondingly, we propose defense solutions by rescheduling the DNN operator-level tensor program and obfuscating the side-channel leakage. Note that although this paper mainly discusses EM side-channel and VTA, the presented experimental observation and the proposed methodologies can be generally extended to other side-channels and hardware platforms.

The main contributions of this work are as follows:

- We comprehensively study the EM side-channel leakage for DNN models deployed with VTA. To the best of our knowledge, this is the first work exploring the association between low-level tensor program code and EM trace, from the adversarial perspective.
- We systematically formulate the mathematical representation of the DNN architecture with EM emanations, following which we further propose security metrics for the DNN models based on the visualized EM characteristics. These metrics can be used to estimate the security level of a DNN architecture against side-channel attacks.
- We present NNReArch, a flexible defense framework for DNN model architectures against EM side-channel attacks. We evaluate the performance of NNReArch with state-of-the-art DNN architectures, and we demonstrate that NNReArch can significantly complex the reverse

engineering attacks, i.e., doubling the attacking efforts with only 3.06% performance overhead.

II. BACKGROUND AND RELATED WORK

A. Versatile Tensor Accelerator (VTA)

VTA is an open-source, generic, and FPGA-specific deep learning acceleration framework [6], consisting of four modules to enable task-level parallelism in a pipelining fashion (TLPP): a "fetch module" that loads the instruction stream from the DRAM, a "load module" loading the input, weight parameters, and intermediate results, a "store module" writing back intermediate results, and the "compute module" accelerates computing. The last one is "compute module" that relies on two kernels, the general matrix multiply (GEMM) kernel for dense linear algebra computations, and the tensor arithmetic logic unit (ALU) for general computing tasks. VTA is supported by TVM [3], a compiler (AutoTVM) scheduling the target deep learning application on a hardware platform. In addition, two techniques, matrix multiplication blocking (MMB) and virtual threading (VT), are used to customize the FPGA execution to further improve the VTA performance. The MMB technology divides large NN operators down to smaller blocks to fit the GEMM kernel, and VT manages the hardware resources of VTA to facilitate simultaneous computing and memory access. More technical details of the VTA can be found in Sec. III.

B. Model Architecture Extraction Attack and Defense

Model architecture extraction has become an emerging threat to the security of DNN. In addition to attacks from the software side [12], different side-channels have been utilized to extract DNN architectures on hardware platforms. Batina *et al.* [13] first demonstrated architecture extraction of multi-layer perception (MLP) using the EM signals from both AVR and ARM processors. In [9], Yu *et al.* applied EM-based attack on an FPGA against a convolutional neural network (CNN). Tian *et al.* [14] utilized an on-chip time-to-digital (TDC) sensor to extract the NN architecture, which can be launched remotely on a multi-tenant FPGA, but with lower resolutions compared to EM signals. Hu *et al.* [15] demonstrated extracting a complete NN architecture using multiple side-channels of GPUs, such as the memory access pattern. However, none of the prior works targets comprehensively studying the tensor programmable accelerator, such as VTA. Also, defense methods are still in their infancy.

C. EM Side-channel

EM side-channel [13] is an effective and contact-less method for extracting sensitive information during the system execution. The instantaneous EM emanation is dependent on the dynamic current [9], as shown in Eq. 1:

$$I_{dyn}(t) = \frac{C \times V_{DD} \times f_{clk} \times D(t)}{2} \quad (1)$$

where C denotes the capacitance of the activated metal nets, $D(t)$ represents the transition rate of the nets (determined by both operations and data), V_{DD} is the voltage supply, and f_{clk}

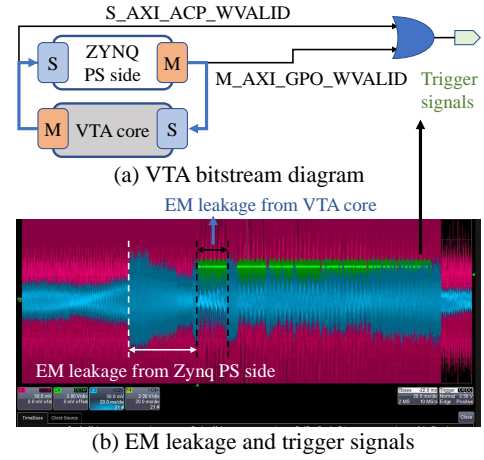(a) VTA bitstream diagram

(b) EM leakage and trigger signals

Fig. 1: Experimental Platform for EM Signal Collection

stands for the execution clock frequency [16]. Therefore, an EM trace from a hardware platform well embodies information for its workload, data, and system computing and communication.

III. ASSOCIATING TENSOR PROGRAM ON VTA WITH EM EMANATION

A. Threat model

We follow the same threat model as in other related side-channel model extraction works [9, 13], with our victim device being an edge FPGA running an VTA accelerator for a pre-trained DNN model. The attacker is able to obtain the EM signals of the victim device with specialized equipment and also knows the model execution status, which can assist in reasoning the architecture of the victim DNN model. We assume a strong attacker, who has sufficient knowledge of the target accelerator, including the configuration of the accelerator (discussed in Sec. III-C2). The executing model architecture is the target for reverse engineering with all the EM traces, accelerator, and platform information.

B. Experimental Platform

We build an experimental platform using PYNQ-Z1 development kit, an SoC with a Xilinx Zynq-7000 device and a dual-core ARM Cortex-A9 processor (PS). To align the EM signals with execution phases for the device characterization purpose, we modify the VTA bitstream as shown in Fig.1 (a). Specifically, we use two signals as the trigger signals to mark the starting and ending of the VTA execution (and therefore the corresponding EM segment): the general-purpose output (GPO) manager write valid of the PS, *M_AXI_GPO_WVALID*, and the accelerator coherence port (ACP) subordinate write valid of the VTA core, *S_AXI_ACP_WVALID*. The *M_AXI_-GPO_WVALID* signal annotates the opcodes and data that have been written into the VTA queues, and *S_AXI_ACP_WVALID* indicates that the VTA core output is valid to be written back to the DRAM of the PYNQ-Z1 system.

Our EM trace collection setup includes an EM Probe PBS2 [17] converting the EM signals into voltage represen-
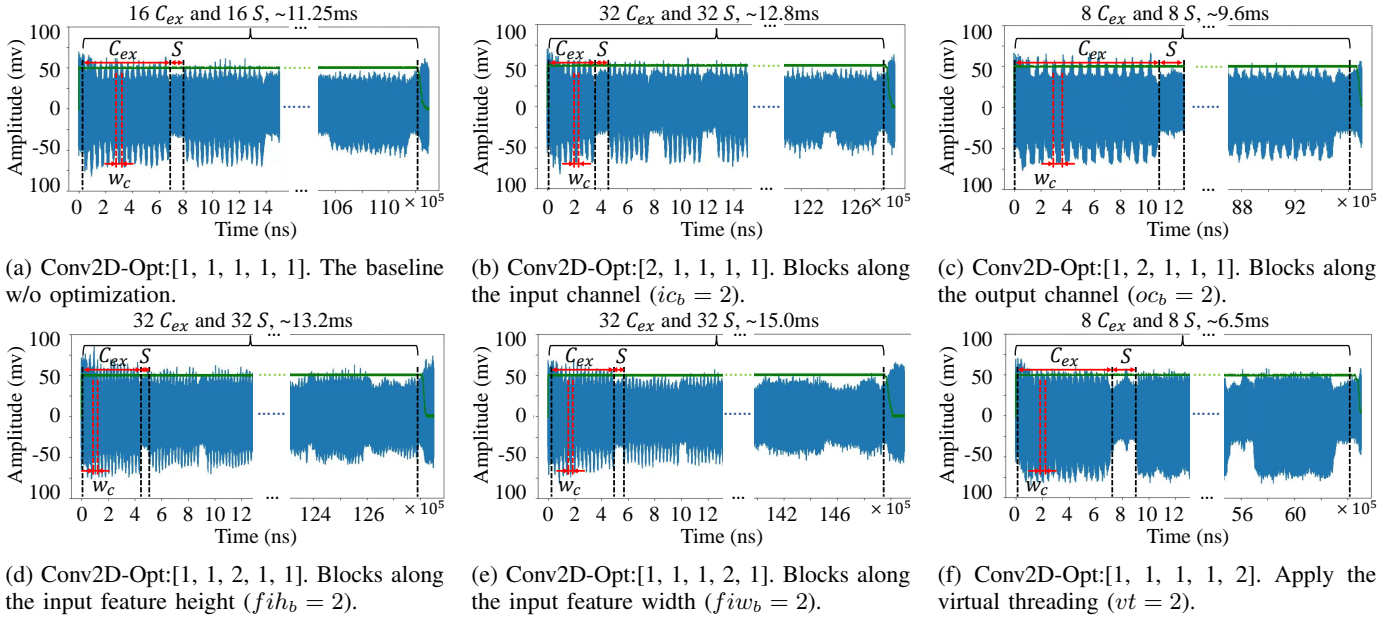
(a) Conv2D-Opt:[1, 1, 1, 1, 1]. The baseline w/o optimization.

(b) Conv2D-Opt:[2, 1, 1, 1, 1]. Blocks along the input channel ($ic_b = 2$).

(c) Conv2D-Opt:[1, 2, 1, 1, 1]. Blocks along the output channel ($oc_b = 2$).

(d) Conv2D-Opt:[1, 1, 2, 1, 1]. Blocks along the input feature height ($fih_b = 2$).

(e) Conv2D-Opt:[1, 1, 1, 2, 1]. Blocks along the input feature width ($fiw_b = 2$).

(f) Conv2D-Opt:[1, 1, 1, 1, 2]. Apply the virtual threading ($vt = 2$).

Fig. 2: We use Conv2D-Wop:[256, 256, 3, 14, 14] as an example to show how the EM leakages change with the Opt-config $[ic_b, oc_b, fih_b, fiw_b, vt]$.

tations, an Aronia AG pre-amplifier, and a Lecroy oscilloscope [18]. An EM trace example with the sampling rate of 1GHz is shown in Fig. 1 (b), which is averaged from 50 measurements with the same inputs. The average pre-processing method helps to make the EM trace stable and filter the measurement noise.

C. Terminology and Definitions

With the EM measurement setup fixed, there are three other factors that jointly determine the EM trace measurement of a VTA: (1) The workload configuration of the current DNN layer's operation, namely **Wop-config**; (2) The global configuration of the VTA-core, namely **VTA-config**; (3) To take advantage of the FPGA parallelism and ARM multi-thread scheduling, operators (resources) for a DNN layer are optimized, whose configuration is denoted as **Opt-config**.

1) Wop-config: For the most commonly used DNN architectures, 2D convolutional layer (Conv2D) is the most critical component to construct the entire model architecture. A Conv2D is specified by the number of input channels (IC), output channels (OC), the kernel size (K), the input feature size (FI), and the output feature size (FO). We follow the typical regulation that assumes the input/output feature is square-shaped. The Conv2D-Wop is therefore $[IC, OC, K, FI, FO]$.

2) VTA-config: A VTA-core is deployed on an FPGA specified by VTA-config, the configuration from [11]. The main computing component GEMM kernel, is designed around a tensor core performing one matrix-matrix operation in each clock cycle. This operator is to implement the product of a 1×16 input and a 16×16 weight matrix. The VTA core employs hardware resources for parallel computation to achieve high performance. The input matrix has dimension of $BATCH \times BLOCK\_IN$, where $BATCH$ indicates how

many feature maps can be implemented in parallel, by the VTA core ($BATCH = 1$ by default), and $BLOCK\_IN$ represents the input channel-parallelism. For example, in our experimental, $BLOCK\_IN$ is set as 16, indicating that 16 input channels can execute in parallel. The weight matrix includes $BLOCK\_IN \times BLOCK\_OUT$ number of weights, where $BLOCK\_OUT$ represents the output channel-parallelism. When $BLOCK\_OUT$ is 16, the VTA can produce results in 16 output channels (output $BATCH \times BLOCK\_OUT$). Another setup of VTA-config is the sizes of on-chip buffers, including input, output, and weight buffer with 32KB, 128KB, and 32KB memory sizes, respectively.

3) Opt-config: Rather than static execution, the VTA can dynamically schedule the execution of Conv2D layers for performance optimization. AutoTVM supports VTA to implement explicit memory latency hiding by the virtual threading (vt) primitive, corresponding to multi-threading of the ARM processor. As our used ARM Cortex A9 dual-core processor allows two threads, the VTA vt can support threads up to 2. To map the matrix multiplication efficiently on a VTA core, TVM can optimally break down large workload as smaller blocks, to achieve computation efficiency within limited hardware resources. There are four scales of blocks associated with this technology: input channel blocks (ic_b), output channel blocks (oc_b), and two input feature map blocks along the height axis (fih_b) and width axis (fiw_b), respectively. We use an Opt-config vector $[ic_b, oc_b, fih_b, fiw_b, vt]$ to represent the optimization setting. For example, a Conv2D-Opt of [2, 2, 2, 2, 2] is applied on a convolution layer with Con2D-Wop: [256, 256, 3, 14, 14]. The scheduler will divide the original convolution layer into several small blocks with workload $Conv2D_b$-Wop:[128, 128, 3, 14, 14] because both the input (ic_b) and output channel (oc_b) blocks are 2. It will also separate
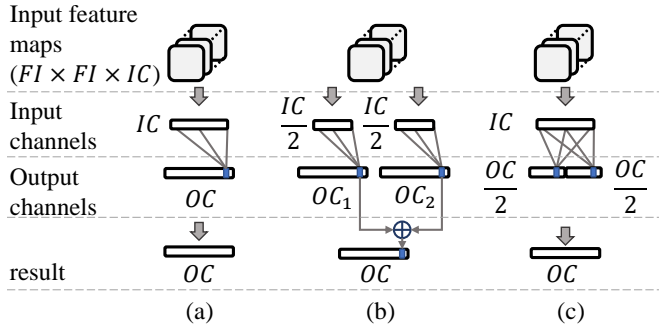
Fig. 3: The scheduler divides a large Conv2D layer down to smaller blocks along the IC and OC axes. (a) w/o optimization. (b) Conv2D-Opt: [2, 1, 1, 1, 1], $ic_b = 2$. (c) Conv2D-Opt: [1, 2, 1, 1, 1], $oc_b = 2$.

the 14×14 input feature map into 7×7 blocks because fih_b and fiw_b are also 2. Moreover, the Opt-config enables dual-threading. In contrast, the non-optimized Opt-config is Conv2D-Opt:[1, 1, 1, 1, 1]. Note that across this paper, we use the default $BATCH$ setting, and the subscript $_b$ is used to represent the detailed value of each parameter out of many possibilities.

D. EM Leakage Observation

In our experiment, we implement a convolutional layer with Conv2D-Wop of [256, 256, 3, 14, 14]. We choose different Opt-configs to understand the impact of optimizations on DNN execution, which is reflected in the EM leakage. Fig. 2 shows the EM traces collected from the basic VTA setting without optimization (Fig. 2a) and five optimized versions (Fig. 2b to 2f). Inspecting these traces, we find a repetitive pattern of a segment of high-frequency activity (continuous execution, C_{ex}) followed by a segment of low-frequency (stalling) activity (S). This pattern repeats M times for the convolutional layer computation. Further, from the beginning C_{ex} segment, we can clearly observe several spikes, the number (N) of which is countable and each of them has approximately the same width (w_c), where $C_{ex} = N \times w_c$. Thus, we can derive a Conv2D EM trace function ($Conv2D_{EM}$) with 2 countable parameters: M and N, and 2 measurable parameters: w_c and S.

$$Conv2D_{EM} = M \times (N \times w_c + S) \qquad (2)$$

When we change the ic_b along the IC and oc_b along the OC of the Opt-config, the computing flow for each axis' blocks is shown in Fig. 3, and their corresponding EM traces are shown in Fig. 2b and Fig 2c. If dividing the Conv2D layer along the IC axis, it will generate ic_b subordinated outputs OC_i, whose summation will derive the final result. When $ic_b = 2$, two paths are scheduled and the execution time of C_{ex} is reduced by almost a half. Blocks along the OC axis has a simpler computational process. The scheduler separates OC output channels into oc_b sections, and the final result is the concatenation of those sub-results. Comparing Fig. 2a with Fig. 2b and Fig. 2c, the IC axis blocking determines M and N, and the OC axis blocking determines M and w_c.

Similarly, blocks along the feature map height and width also affect M and w_c of the EM trace, as shown in Fig. 2d and Fig. 2e, although no obvious difference between these two EM traces can be observed. Following our measurement results, blocking along the width of the feature map (fiw_b) induces a longer S. Besides, we applied the virtual threading method, which accelerates the operator by hiding the DRAM memory access latency and enables the TLPP of VTA as mentioned in Sec.II-A. As shown in Fig. 2f, this configuration shortens the execution time of the entire Conv2D layer compared with the baseline by reducing the M.

From the EM leakage observation, we can draw the following conclusions: (1) M is a function of IC, ic_b, OC, oc_b, FO, FI, fih_b, fiw_b, and vt; (2) N is a function of IC, ic_b; and (3) w_c is a function of OC, oc_b, FI, fih_b, and fiw_b.

E. Low-level Program Code Analysis

Visually inspecting the EM traces derives general association between the EM pattern and the two configurations, Wop-config and Opt-config. To comprehensively understand the execution impacts on the EM leakage, we look into the low-level code structure shown in Fig 4. Since the Tensor ALU operators of a Conv2D layer have low-arithmetic intensity and therefore do not emanate high EM leakage, we focus on the GEMM operator [6]. The GEMM code is composed by many nested loops of operations, corresponding to the repetitive EM pattern shown in Fig 2. We extract three parts (part 1 to 3 as shown in Fig 4) related to the $Conv2D_{EM}$ function parameters, M, N, and w_c, respectively. In Part 1, there are four outer loops related to M, and their ranges indicate the blocking parameters: ic_b, fih_b, fiw_b and vt. Note that the part 1 program is the most outer loop, the function of M is also determined by several other parameters, as defined in Eq. 3:

$$M = \frac{IC \times ic_b \times FO \times fih_b \times fiw_b}{BLOCK\_OUT \times FI \times oc_b \times vt} \qquad (3)$$

It is straightforward to determine N from the range of `ic.outer` of the Part 2 code:

$$N = \frac{IC}{BLOCK\_IN \times ic_b} \qquad (4)$$

Different from M and N that are discrete (integer) numbers, w_c is associated with the execution time. Hence, without knowing the exact function, we can only leverage the Part 3 code to determine which parameters affect its quantity. In the first `cthread.s_1` loop, if its range is larger than 1, it will enable the TLPP. Our experiments suggest that the range of `dx` is equal to K, i.e., the kernel size. If $oc_b = 1$, the range of `dy` is also 1, otherwise it is K. The range of j is a function of FI, fiw_b, and FO. Putting all these clues together, we assume function $g(\cdot)$ can obtain w_c from low-level tensor program code in Eq. 5.

$$w_c = g(K, FI, fiw_b, FO, vt, f_{ex}, II) \qquad (5)$$

The TLPP is configured by vt, II denotes the initiation interval for the pipeline, and f_{ex} is the executing frequency of the VTA core, which is $100MHz$ in this paper.

```
attr = {"from_legacy_te_schedule": True, "global_symbol": "main", "tir.noalias": True}
buffers = {res: Buffer(res_2: Pointer(int8), int8, [1, 16, 14, 14, 1, 16], []),
           data: Buffer(data_2: Pointer(int8), int8, [1, 16, 14, 14, 1, 16], []),
           kernel: Buffer(kernel_2: Pointer(int8), int8, [16, 16, 3, 3, 16, 16], [])}
```
— Describe the Conv2D workload

```
buffer_map = {data_1: data, kernel_1: kernel, res_1: res} {                          Part 1
    for (i1.outer.outer: int32, 0, 16) {
        for (i2.outer: int32, 0, 2) {          i2.outer appear if enable fih_b
            for (i3.outer: int32, 0, 2) {      i3.outer appear if enable fiw_b       Related to M
                for (cthread.s: int32, 0, 2) { The range of cthread.s > 1 if enable vt
                    load VTAPushGEMMOp stream to VTA queue...}}}
```

```
        for (ic.outer: int32, 0, 16) {                                              Part 2
            Calculate the DRAM memory address and load data by VTALoadBuffer2D...}   Related to N
                ... may have additional same code blocks depending on the cthread.s range
```

```
        for (cthread.s_1: int32, 0, 2) {                                            Part 3
            VTAUopLoopBegin ...
            for (dy: int32, 0, 3) {        The range of cthread.s_1 > 1 if enable vt
                for (dx: int32, 0, 3) {    dy appear if enable oc_b    Related to w_c
                    for (j: int32, 0, 14) {
                        Execute the computation...}}}
        ... VTAUopLoopEnd}}}}}
```

Fig. 4: Low-level code summary for the optimizable and high-arithmetic intensity GEMM operator, which constructs the Conv2D layer with low-arithmetic intensity ALU.

(a) $Conv2D_o$-Wop:[256, 256, 3, 14, 14], $Conv2D_o$-Opt:[1, 2, 1, 1, 2].

(b) $Conv2D_t$-Wop:[128, 128, 3, 28, 28], $Conv2D_t$-Opt:[1, 4, 2, 2, 2].
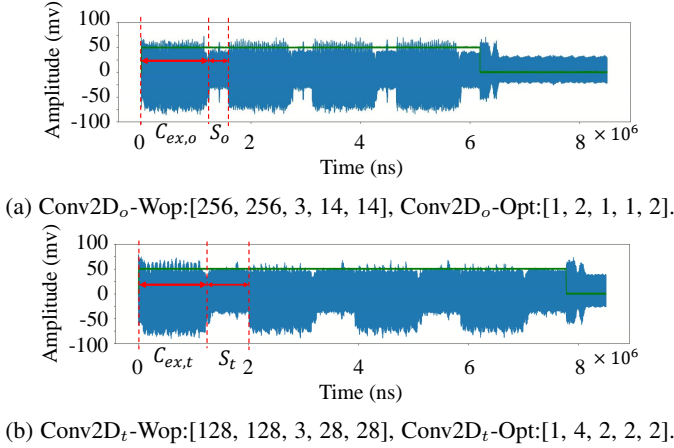
Fig. 5: Conv2D EM obfuscation example. We use the subscript o to represent the original Conv2D layer, and subscript t is the target obfuscation Conv2D layer.

F. EM Obfuscation

With the EM leakage characterization and the low-level code analysis, we propose to obfuscate the EM trace by scheduling the tensor program. As a proof of concept, we implement two different Conv2D layers: one $Conv2D_o$ with the Wop-config of [256, 256, 3, 14, 14] and Opt-config of [1, 2, 1, 1, 2]; the other $Conv2D_t$ with the Wop-config of [128, 128, 3, 28, 28] and Opt-config of [1, 4, 2, 2, 2]. Inspecting their EM traces in Fig. 5, we notice these two layers can generate similar C_{ex}, because their GEMM operators have equal counting result (OpC_{GEMM}) derived by Eq. 6.

$$OpC_{GEMM} = \frac{IC \times OC \times K^2 \times FO^2}{BLOCK\_IN \times BLOCK\_OUT} \quad (6)$$

However, the stall time is different between these two settings (S_o and S_t). Such difference can be canceled by adding **pause opcode** to delay $\sim 0.35ms$ in every S_o, so that $S'_o = S_t$. As a result, these two EM traces become in-distinguishable(i.e., unable to determine which setting is in effect).

Generally, the goal of EM obfuscation for a Conv2D layer is to find different configurations resulting in the same operation counts as the original one, following Eq. 6. Specifically, this equation can assist us to find a target Wop-config $Conv2D_t - Wop$, for which there exists an Opt-config $Conv2D_t - Opt$ satisfying $M_t = M_o$, $K_o = K_t$, and has a longer execution time. Then we can derive $\Delta S = S_t - S_o$ by measurement, and apply it to the original workload to mimic it as the target workload.

IV. SECURITY METRICS AND OPTIMIZATION PROGRAM FOR VTA IMPLEMENTATION

This section introduces NNReArch, which utilizes Opt-config and EM obfuscation to mitigate the EM leakage of the DNN model. For an attacker to reverse-engineer the victim NN architecture implemented on VTA, s/he needs to derive VTA-config and Opt-config. If Opt-config is fixed, such as Conv2D-Opt of [1, 1, 1, 1, 1], then the victim architecture is easy to extract. Thus, a primary idea of mitigating the EM side-channel leakage is to increase the searching space of the Opt-config for each Conv2D layer.

A designer can formulate the scheduling of DNN execution as an optimization problem. For a given neural network architecture NN, we can extract a set of workload expression E that executes on a target acceleration device. Then, for a given workload $e \in E$, we can implement it with many different functionally equivalent low-level program codes inducing different EM traces, as observed in Sec. III-D and III-E. Therefore, each workload could have multiple equivalent schedules, i.e., Opt-config. We use Ps_e to denote the possible schedule space for e. For example, in VGG-19, there are 9 types of Conv2D layers with different Wop-configs, each of whic is denoted as $e_i, i \in [1, 9]$ and has a set of Opt-config Ps_{e_i}.

A. Security Metrics

When considering confidentiality of a NN, brute force attack is the most generic method and its complexity can be represented by the size of its searching space (SS_{NN} defined in Eq. 10). The attacker normally progresses sequentially, i.e., from the first Conv2D layer to the following layers, since s/he has to utilize the results (e.g., dimensions) of the previous layers. IC and FI of the first layer can be directly observed from the input image and global configuration of the accelerator. For the Wop-config of each Conv2D layer, the adversary could build a library of combinations of Wop-config and Opt-config, and then estimate their EM trace patterns. The candidate with high similarity to the observed EM trace of the target NN could be considered as the correct hypothesis with high confidence.

1) Search space for individual Conv2D layers: For a Conv2D layer, generally we assume IC, FI are derived from the previous Conv2D layer's OC, FO, or the Pooling layer. Other parameters, including $\{K, FO, OC, fih_b, fiw_b, ic_b, oc_b, vt\}$, remain to be discovered. Some of these parameters follow some conventions that can be used as hints for guessing. **Hint 1:** The K of the 1st Conv2D layer might be 3, 5, or 7, and the K of the rest Conv2D layers might be 1 or 3. **Hint 2:** FO depends on the FI and the stride of the kernel, which is normally 1 or 2. When FI is smaller than 8, the stride will be 1. **Hint 3:** OC depends on IC and $BLOCK\_IN$, where IC is expected as a multiple of $BLOCK\_IN$. An exception is that the 1st Conv2D layer usually has an IC smaller than $BLOCK\_IN$, so the VTA will convert it to $IC = BLOCK\_IN$ with dummy input channels. If representing the relationship between OC and IC as $OC = f_{oc} \times IC$, then $f_{oc} \in \{\frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$. Note that $f_{oc} = \frac{1}{4}$ and $\frac{1}{2}$ do not happen when $IC = BLOCK\_IN$, and $f_{oc} = \frac{1}{4}$ do not occur when $IC = 2 \times BLOCK\_IN$. Hence, the searching space (SS) for K, FO, OC are

$$
\begin{aligned}
SS_K &= \begin{cases} 3 & , i = 1 \\ 2 & , i > 1 \end{cases} \\
SS_{FO} &= \begin{cases} 1 & , FI < 8 \\ 2 & , Otherwise \end{cases} \\
SS_{OC} &= \begin{cases} 3 & , IC = BLOCK\_IN \\ 4 & , IC = 2 \times BLOCK\_IN \\ 5 & , Otherwise \end{cases}
\end{aligned} \tag{7}
$$

Following Eq. 7, an attacker can formulate the searching space for the potential Wop-conifgs of $Conv2D_i$. For a specific Wop-config, the size of its Opt-config searching space can be derived from Eq. 8, again we use the subscript $_b$ to represent the detailed value of each parameter out of many possibilities.

$$
\begin{cases}
SS_{fih_b} = SS_{fiw_b} = \left\lceil log_2 \frac{FI}{4} \right\rceil \\
SS_{ic_b} = \left\lceil log_2 \frac{IC}{BLOCK\_IN} \right\rceil + 1 \\
SS_{oc_b|OC} = \left\lceil log_2 \frac{OC}{BLOCK\_OUT} \right\rceil + 1 \\
SS_{vt} = \max(vt)
\end{cases} \tag{8}
$$

Assuming $SS_c = SS_K \times SS_{FO} \times SS_{fih_b} \times SS_{fiw_b} \times SS_{ic_b} \times SS_{vt}$, the searching space SS_{Conv2D_i} of $Conv2D_i$ can be derived using Eq. 9:

$$
SS_{Conv2D_i} = SS_c \times \sum_{OC \in \Omega_{OC}} SS_{oc_b|OC} \tag{9}
$$

where Ω_{OC} denotes the corresponding values in the searching space of OC. For example, as shown in Tab. I, the 2nd Conv2D layer has $IC = 64$, thus $SS_{OC} = 5$ ($BLOCK\_IN$ and $BLOCK\_OUT$ are set as 16 in Sec. III-C2), and the specific values $\Omega_{OC} = \{16, 32, 64, 128, 256\}$ with possible $SS_{oc_b|OC}$ is 1, 2, 3, 4, 5, respectively. Considering the derivation $SS_c = 864$ for the 2nd Conv2D layer in VGG-19, thus the searching space for this layer is $SS_{Conv2D_2} = 12960$.

2) Searching space for the entire neural network: An attacker needs to iterate all possible Wop-configs and their Opt-configs, to compare the guessed EM leakage with the obtained NN EM leakage starting from the 1st Conv2D layer. We can narrow the search space based on two facts: (1) Multiple Conv2D layers exist in one NN with the same Wop-config; (2) A specific Opt-config may be suitable for different Wop-configs to execute effectively. A strong and practical scenario is that an attacker only tries to utilize the prior knowledge before s/he really has to search the entire space. For example, an attacker is always applying the recovered Wop-config and Opt-config of the previous layer first. In other words, s/he has to checks all these already-discovered Wop-config (SG_{wop}) and Opt-config sets (SG_{opt}), only if the previous knowledge is not working. When both tests fail, s/he iterates other possible configurations. We show an example in Tab. I column **AutoTVM Opt-config**, where e_3 and e_4 employ the same Opt-config. We calculate the searching space for each Conv2D layers and derive the NN searching space SS_{NN}, which also represents the basic security level.

$$
SS_{NN} = \sum_{i=1}^{l} SS_{Conv2D_i} \tag{10}
$$

3) EM obfuscation scheme: For defense, one can use EM obfuscation. In details, a designer can follow Eq. 6 to calculate the OpC_{GEMM} in each Conv2D layer and find all potential EM obfuscation Wop-configs and Opt-config, using Eq. 3 and 4. According the combination theory, the best choice of layers to apply EM obfuscation is $\left\lceil \frac{l}{2} \right\rceil$, where l is the number total layers. Therefore, the designer can randomly select 8 layers out of the 16 layers of VGG-19 to obfuscate. Consequently, the searching space of brute force attack will be increased to a huge number, since the Conv2D configurations reflected in the EM trace does not help with reverse engineering at all. Further, as the Eq. 2 shows, S can be changed (elongated), which will affect the similarity calculation in brute force attack. It is hard for an attacker to find a unique correct Wop-config for the current layer. If they ignore S and only compare M, N, and w_c, it will induce many possible Wop-configs.

B. NNReArch Framework

NNReArch is an automated tensor program generator that can mitigate the DNN-architecture-relevant EM side-channel

TABLE I: VGG-19 under *balance mode* with different scheduling

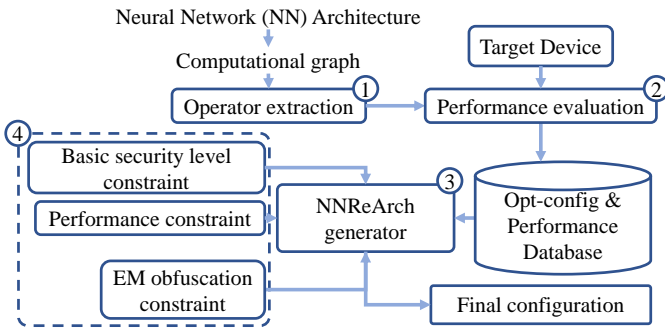| e_i	Conv2D$_i$	IC	FI	Independent SS_{Conv2D_i}	Wop-config Ground Truth	AutoTVM Opt-config	AutoTVM SS_{Conv2D_i}	NNReArch Opt-config	NNReArch SS_{Conv2D_i}
1	1	16	224	2592	[16, 64, 3, 224, 224]	[1, 2, 32, 4, 2]	2592	[1, 2, 32, 4, 2]	2592
2	2	64	224	12960	[64, 64, 3, 224, 224]	[1, 1, 8, 8, 2]	12960	[1, 1, 8, 8, 2]	12960
3	3	64	112	9000	[64, 128, 3, 112, 112]	[1, 2, 8, 4, 2]	9000	[1, 2, 8, 4, 2]	9000
4	4	128	112	16000	[128, 128, 3, 112, 112]	[1, 2, 8, 4, 2]	20	[1, 1, 4, 4, 2]	16000
5	5	128	56	10240	[128, 256, 3, 56, 56]	[1, 2, 4, 2, 2]	10240	[1, 2, 4, 2, 2]	10240
6	6	256	56	16000	[256, 256, 3, 56, 56]	[1, 2, 4, 2, 2]	20	[1, 2, 8, 2, 2]	16000
	7						1	[1, 2, 4, 2, 2]	16000
	8						1	[1, 1, 8, 2, 2]	16000
7	9	256	28	9000	[256, 512, 3, 28, 28]	[1, 1, 1, 1, 2]	9000	[1, 1, 1, 1, 2]	9000
8	10	512	28	12960	[512, 512, 3, 28, 28]	[1, 4, 4, 1, 2]	12960	[1, 1, 4, 1, 2]	12960
	11						1	[1, 2, 4, 1, 2]	12960
	12						1	[1, 4, 4, 1, 2]	12960
9	13	512	14	5760	[512, 512, 3, 14, 14]	[1, 4, 1, 1, 2]	5760	[1, 4, 1, 1, 2]	5760
	14						1	[1, 2, 1, 1, 2]	5760
	15						1	[1, 2, 1, 2, 2]	5760
	16						1	[1, 1, 1, 2, 2]	5760
SS_{NN}							62559		169712
All Conv2D Execution Time (ms)							1670.02		1721.19

Fig. 6: NNReArch framework design overview.

TABLE II: All possible workload configuration of Conv2D layers in VGG-19 applied in NNReArch to build the Opt-config & Performance Database (Fig. 6).

e_i	Wop-config	Ps_{e_i} size	NO. of Usage
1	[16, 64, 3, 224, 224]	61	1
2	[64, 64, 3, 224, 224]	183	1
3	[64, 128, 3, 112, 112]	219	1
4	[128, 128, 3, 112, 112]	292	1
5	[128, 256, 3, 56, 56]	316	1
6	[256, 256, 3, 56, 56]	395	3
7	[256, 512, 3, 28, 28]	315	1
8	[512, 512, 3, 28, 28]	378	3
9	[512, 512, 3, 14, 14]	216	4

leakage from the target device. Fig. 6 shows the workflow of NNReArch. In step ①, we implement the operator extraction. The input is an abstracted DAG NN architecture, and the output is the operator expression list, as shown in Tab. II column e_i. Each of these operators is for certain Conv2D layers with different configurations. Through the TVM compiler we can generate different Opt-configs for a specific e_i workload, as shown in the column **Ps$_{e_i}$ size**, the number of compilable Opt-configs (Ps_{e_i}). We evaluate their performance on the target device in step ②. We record all Ps_{e_i} and their corresponding execution time in an "Opt-config & Performance Database". Step ③ denotes the proposed NNReArch, where the user may apply three constraints shown in ④ to generate the corresponding

low-level NN execution codes. In addition, they can also trade off the security and performance through these constraints. Two Opt-config selecting modes are provided for users: *balance mode* and *secure mode*. In *balance mode*, NNReArch selects the high-performance Opt-config while ensuring the architecture confidentiality level. Specifically, the generator will first select the high-performance Opt-config. *Secure mode* prioritizes security, by randomly choosing Opt-configs and a certain number of layers to apply the EM obfuscation, in order to significantly enlarging searching space. In Fig. 6, the "Basic security level constraint" is related to maximizing the SS_{NN} in both modes. The "Performance constraint" controls the *balance mode* to satisfy the performance requirement, i.e., runtime overhead. The "EM obfuscation constraint" is customized by the user, who only needs to provide the number of layers to obfuscate, and NNReArch will generate low-level candidate code to satisfy all constraints.

V. EVALUATION AND DISCUSSION

In this section, we evaluate the performance of NNReArch, and compare it with AutoTVM, using the most commonly utilized DNN architectures, including VGG-16, VGG-19, ResNet-18, and ResNet-34. For sake of clarity, we list all possible workload configuration (i.e., ground truth) of the Conv2D layers of VGG-19 in Tab. II. For example, "$e_i = 1$" represents the 1st type of Conv2D layer, and its corresponding "NO. of usage = 1" means that this layer type is only used once in VGG-19. From the brute force attack perspective, if given the correct IC and FI, the search space SS_{Conv2D_1} of $Conv2D\_1$ is 2592. Therefore, the attacker will have to iterate entire SS_{Conv2D_1}, to find out the best matching workload.

We present the technical detail of deploying NNReArch on VGG-19 as an example, and illustrate the experimental results of all other DNN architectures in Fig. 7.

A. Applying NNReArch on VGG-19

We firstly apply the *balance mode* of NNReArch on VGG-19, which considers the tradeoff between security and performance, and the results are shown in Tab. I. Specifically,

TABLE III: NNReArch applied on 8 Conv2D layers with *secure mode* (EM obfuscation)

e_i	$Conv2D_i$	NNReArch Wop-config	NNReArch Opt-config
1	1	[16, 64, 3, 224, 224]	[1, 2, 32, 4, 2]
2	2	[64, 64, 3, 224, 224]	[1, 1, 8, 8, 2]
3	3	[64, 128, 3, 112, 112]	[1, 2, 8, 4, 2]
4	4	[128, 128, 3, 112, 112]	[1, 1, 4, 4, 2]
5	5	[128, 256, 3, 56, 56]	[1, 2, 4, 2, 2]
6	6	[256, 64, 3, 56, 112][1]	[1, 2, 8, 2, 2]
	7	[64, 256, 3, 112, 112][2]	[1, 2, 4, 2, 2]
	8	[256, 256, 3, 112, 56][3]	[1, 1, 8, 2, 2]
7	9	[256, 512, 3, 28, 28]	[1, 1, 1, 1, 2]
8	10	[512, 512, 3, 28, 28]	[1, 1, 4, 1, 2]
	11	[512, 512, 3, 28, 28]	[1, 2, 4, 1, 2]
	12	[512, 2048, 3, 28, 14][4]	[1, 4, 4, 1, 2]
9	13	[2048, 512, 3, 7, 7][5]	[1, 4, 1, 1, 2]
	14	[512, 2048, 3, 7, 7][6]	[1, 2, 1, 1, 2]
	15	[2048, 512, 3, 7, 7][7]	[1, 2, 1, 2, 2]
	16	[512, 512, 3, 7, 14][8]	[1, 1, 1, 2, 2]
All Conv2D Execution Time (ms)			1927.59

we apply the security constraint as "maximizing the Sec_b" and the performance constraint as "shortest execution time". Note that this evaluation does not include the EM obfuscation. The performance of applying NNReArch is mainly shown in column **NNReArch Opt-config** in Tab. I. The total execution time of all VGG-19 Conv2D layers is 1721.19ms. Compared with the original performance using "AutoTVM" scheduling (column **AutoTVM Opt-config** in Tab. I), the deployment of NNReArch only incurs 3.06% performance overhead. In contrast, the searching space (SS_{Conv2D_i}) is significantly increased, with $SS_{NN} = 169712$. Thus, in terms of the *balance mode*, applying NNReArch (w/o EM obfuscation) increases the difficulty of DNN architecture extraction for about 2.71 times.

We further apply the *secure mode* of NNReArch (i.e., with EM obfuscation) to prioritize the DNN model architecture confidentiality. Specifically, we select 8 layers of VGG-19 to schedule their tensor program, making them to have the same EM trace characteristics. The Opt-config setting of the obfuscated workload are listed in Tab. III, and here we use superscripts (1,2,...,8) to annotate the obfuscated layers. Compared with the ground truth configuration (column **Wop-config Ground Truth** in Tab. I), the obfuscated EM traces will lead the reverse engineering attack to wrong workload. For example, the EM obfuscation breaks the performing divergence of the EM leakage on different convolution layers, when the attacker reasons each Conv2D layer, i.e., s/he will derive wrong workloads that negatively affect the guess on followed layers, disturbing the consistency of adjacent Conv2D layers. As a result, when the exteriors of different Wop-configs have the same patterns, attackers have to handle a huge amount of puzzles and guesses. Thus the searching space and attacking effort will increase massively.

As discussed in Sec. III-F, the EM obfuscation needs to add pause opcodes to obfuscate the stall time, to make the obfuscated layer performing almost the same as the "imitated layer", which may brings performance loss. In our experimental evaluation on VGG-19 shown in Tab. III, the obfuscated

DNN model consumes 1927.59 ms to execute all Conv2D layers, which has an approximate 15.42% performance overhead compared with the AutoTVM.

B. Discussion: security and performance trade-off

(a) Security level measurement
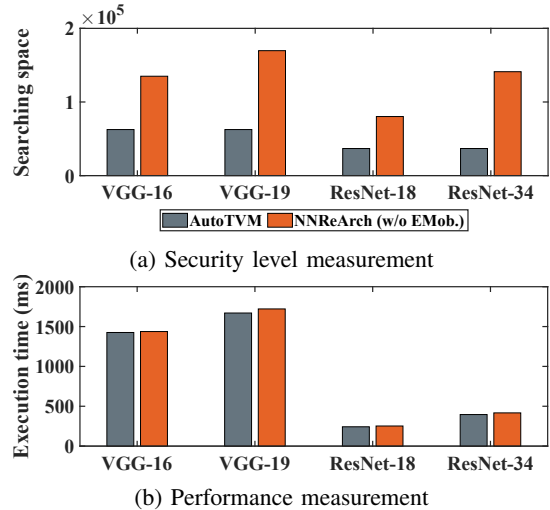

(b) Performance measurement

Fig. 7: Performance evaluation and comparison between AutoTVM and the NNReArch *balance mode* of NNReArch.

Since most existing DNN development frameworks (including both open-source and commercial) still target at performance, therefore, we evaluate the *balance mode* of NNReArch on the popular DNN architectures from VGG and ResNet to draw generic conclusions. We first compare the searching space (SS_{NN}) of DNNs from the same family. For example, VGG-16 and VGG-19 are constructed by the same Conv2D layer types, but only with different number of layers of the same Wop-config. Therefore, the attacking difficulty for DNNs in the same family is almost equal using AutoTVM. As affirmed in Fig. 7a, the searching space is the same for VGG-16 and VGG-19, as well as for ResNet-18 and ResNet-34. This indicates a vulnerability of these performance-only optimization frameworks, i.e., using a larger DNN model does not make it more challenging to reverse engineer the model architecture. In contrast, the proposed NNReArch framework constructively leverages the model size to maximizes the searching space of each Conv2D layer, making the architecture of deeper DNNs more secure, as shown in Fig. 7a. Fig. 7b illustrates the performance comparison between AutoTVM and NNReArch, which demonstrates that NNReArch only incurs trivial execution time overhead on all evaluated DNN architectures compared with AutoTVM.

For the *secure mode* of NNReArch, we demonstrate that a carefully crafted DNN model (e.g., the one in Tab. III), can significantly challenge the model extraction attacks with relatively higher performance loss (15%). Moreover, the *secure mode* enables the designer to either randomly choose Conv2D layers for EM obfuscation, or apply unexpected stall time to break the association between the EM side-channel leakage and workload configuration, thus providing more flexibility to the DNN model security enhancement.

VI. Conclusion

In this paper, we study the association between DNN model architecture configuration and EM side-channel leakage. Using an open-source deep-learning accelerator VTA as our experimental platform, we discover the low-level code causes of the EM side-channel-enabled DNN architecture reverse engineering attacks. Furthermore, we present NNReArch, a DNN model architecture defense framework against side-channel attacks. Enabling flexible DNN configuration between performance and security, NNReArch integrates two modes, *balance mode* that targets at increasing the searching space of DNN model architectures, and *secure mode* that employs EM obfuscation to cancel the difference between different model layers. Different from the existing solutions, the proposed framework is built on popular open-source DNN compilation tools, VTA, making it a generic defense method.

References

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[2] Dpu for convolutional neural network. https://www.xilinx.com/products/intellectual-property/dpu.html.

[3] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.

[4] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[5] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "Dadiannao: A neural network supercomputer," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 73–88, 2016.

[6] T. Moreau, T. Chen, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Vta: an open hardware-software stack for deep learning," *arXiv preprint arXiv:1807.04188*, 2018.

[7] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 295–300.

[8] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, "{Deep-Dup}: An adversarial weight duplication attack framework to crush deep neural network in {Multi-Tenant}{FPGA}," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1919–1936.

[9] 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.

[10] T. Zhou, Y. Zhang, S. Duan, Y. Luo, and X. Xu, "Deep neural network security from a hardware perspective," in *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2021, pp. 1–6.

[11] T. Chen, "Pynq-z1 tvm-vta core configuration." [Online]. Available: https://github.com/apache/tvm-vta/blob/master/config/pynq_sample.json

[12] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1345–1362.

[13] 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.

[14] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "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)*. IEEE, 2021, pp. 242–246.

[15] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

[16] J. H. Anderson and F. N. Najm, "Power estimation techniques for fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, pp. 1015–1027, 2004.

[17] "Aaronia pbs2 e & h near field probe set sniffer dc to 6ghz with emc preamplifier," https://instrumentcenter.eu/products/emc-products/probes/aaronia-pbs2-e-h-near-field-probe-set-sniffer-dc-to-6ghz-with-emc-preamplifier, (Accessed on 06/06/2021).

[18] "Teledyne lecroy - oscilloscope," https://teledynelecroy.com/oscilloscope/, (Accessed on 06/08/2021).