usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION

SoK: Neural Network Extraction Through Physical Side Channels

Péter Horváth, Dirk Lauret, Zhuoran Liu, and Lejla Batina, *Radboud University*

https://www.usenix.org/conference/usenixsecurity24/presentation/horvath

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

SoK: Neural Network Extraction Through Physical Side Channels

Péter Horváth* *Radboud University*

Dirk Lauret* *Radboud University*

Zhuoran Liu* *Radboud University*

Lejla Batina *Radboud University*

Abstract

Deep Neural Networks (DNNs) are widely used in various applications and are typically deployed on hardware accelerators. Physical Side-Channel Analysis (SCA) on DNN implementations is getting more attention from both industry and academia because of the potential to severely jeopardize the confidentiality of DNN Intellectual Property (IP) and the data privacy of end users. Current physical SCA attacks on DNNs are highly platform dependent and employ distinct threat models for different attack objectives and analysis tools, necessitating a general revision of attack methodology and assumptions. To this end, we provide a taxonomy of previous physical SCA attacks on DNNs and systematize findings toward *model extraction* and *input recovery*. Specifically, we discuss the dependencies of threat models on attack objectives and analysis methods, for which we present a novel systematic attack framework composed of fundamental stages derived from various attacks. Following the framework, we provide an in-depth analysis of common SCA attacks for each attack objective and reveal practical limitations, validated by experiments on a state-of-the-art commercial DNN accelerator. Based on our findings, we identify challenges and suggest future directions.

1 Introduction

Deep Neural Networks (DNNs) have been widely deployed in various applications, e.g., robot vision [138], conversational AI chatbots [96, 123], machine translations [111], and AI-assisted medical imaging [77]. The effectiveness of DNNs fuels their widespread deployment, rendering them prevalent in production. The data-driven nature of DNNs necessitates their dependency on the availability of extensive datasets and computational resources, which demands dedicated hardware accelerators, for which cloud and edge computing are typically being leveraged [3, 8]. However, various applications that are overly reliant on DNNs also increases the adversary's incentive to look at the vulnerabilities of DNN implementations.

Vulnerabilities of DNN implementations have been studied in Privacy-Preserving Machine Learning (PPML) research [7, 46, 88], where training and inference are implemented on hardware platforms from parties that do not trust each other. Current research on PPML is partly conducted at the software level. On the one hand, the data provider worries about data privacy, concerning leakage of private data to the party that conducts the computation. For example, the input to DNN models could be considered as privacy-sensitive, e.g., input to facial-recognition-enabled CCTV cameras [15], medical images locating diseases in a preliminary stadium [103], or a user's prompt to an online conversational AI [96]. Examples of attacks compromising the data privacy of end users are membership inference [108] and attribute inference [131] attacks. On the other hand, the service provider that owns DNN models has concerns that malicious users will steal the Intellectual Property (IP) of DNNs, i.e., the architecture and trained parameters, especially when deploying them to edge devices [11]. One representative attack is *model extraction*, where the adversary can rebuild the model by querying the DNN [66, 119]. Common ways to combat those threats for PPML include differentially-private machine learning [35, 62], secure multi-party computation [67], and homomorphic encryption [100].

Physical SCA attacks pose a substantial threat in the PPML scenario, where they have a different threat model than software-level attacks, hinting that guarantees provided by current defenses against software-based attacks may not hold. Dedicated hardware is being designed and optimized for DNN inference, and widely used in production environments [4, 8]. Physical SCA attacks on DNNs assume that the adversary is in the vicinity of the hardware device that runs DNN models and could collect traces that represent the device's physical properties, such as electromagnetic (EM) leakage or power consumption. Figure 1 provides an illustrative example showing the SCA setup when a DNN is running on hardware. Input data [12, 125], hyperparameters [81], and parameters [41] can

*Authors are listed in alphabetical order.

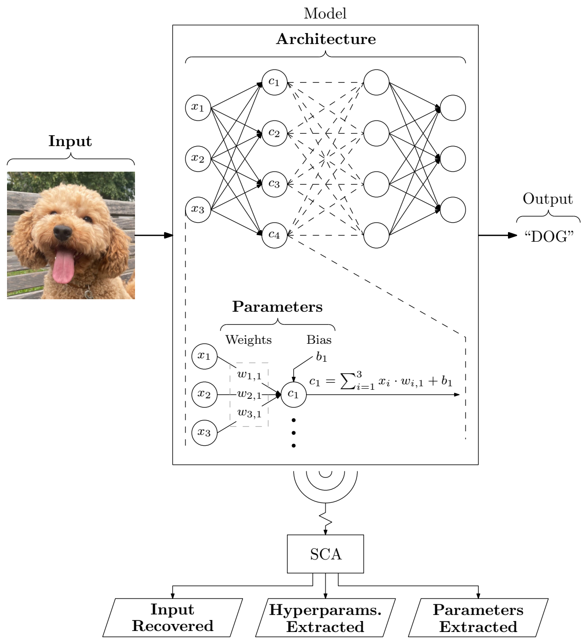

Figure 1: Illustration of physical SCA on a Multi-Layer Perception (MLP), where the model is running on hardware for image classification. In the first layer of the MLP, bias b_1 is added to the weighted sum of all input points (x_i) with weights ($w_{i,1}$) to get an intermediate value c_1. Under a specific threat model, intermediate values can be exploited in the trace analysis to mount an SCA attack to extract inputs or models.

be extracted by analyzing the collected traces, rendering a practical threat. Sometimes, the adversary does not even need to have any information about the model [11]. It indicates that anyone in the physical proximity of the hardware could be considered a threat, which is a valid hypothesis for adversaries being providers of cloud computing or end users of edge devices. For example, a hospital employee can install an EM probe on a portable ultrasound machine [122] and measure the leakage from the chip to steal the DNN models.

Extensive research has been conducted in the area of physical SCA on DNNs over the past few years. Current SCA attacks are highly platform dependent and employ distinct threat models for different attack objectives and analysis tools, demanding a general revision of attack methodology and assumptions. In addition, existing surveys do not consider the fundamental limitations introduced by model extraction through physical side channels [79], but rather focus on the wide scope of model extraction and discuss approaches only at a high level [94].

In this paper, we revisit previous SCA attacks on DNN implementations and provide a taxonomy where we systematize findings toward model extraction and input recovery. Specifically, we discuss the dependencies of threat models on attack objectives and analysis methods, motivated by which we also present a novel systematic attack framework composed of fundamental stages derived from various attacks. Furthermore, we provide an in-depth analysis of common SCA attacks for each attack objective and discuss uncovered potential practice limitations, validated by experiments on a state-of-the-art commercial Xilinx FPGA DNN accelerator.

Specifically, this SoK paper makes the following contributions:

- We taxonomize SCA attacks on DNNs that focus on extracting the architecture, parameters, and inputs exploiting physical side-channel leakages. By breaking down the attack into different stages, we provide a framework that can be used to analyze and categorize all attacks using side channels for model extraction and input recovery.
- Following our framework, we provide in-depth descriptions and comparisons of SCA attacks, summarize the common knowledge, and systematize the threat model.
- We demonstrate that strict assumptions that current approaches make for architecture extraction, can be relaxed in some cases. This results in a larger search space for architectural properties.
- We provide an analysis with simulations demonstrating that the current approaches for parameter extraction of DNNs are not efficient due to error propagation to different layers.
- We present a novel input recovery approach, in which attributes of the input can be recovered without any knowledge of the deployed DNN.
- We identify future challenges of physical SCA attacks based on our framework and edge-case analysis. We suggest future directions regarding hardware developments, physical-SCA-based attacks, and countermeasures.

The rest of the paper is structured as follows. We start the paper by providing background information on DNNs (Section 2) and SCA (Section 3). Then we present the motivation towards model and input recovery attacks, together with the generic threat model and our framework for current approaches (Section 4). We discuss existing approaches towards *architecture extraction* (Section 5), *parameter extraction* (Section 6), and *input recovery* (Section 7). Next, we provide an outlook on the development of SCA, identify new potential avenues for model and input recovery (Section 8). Finally, we discuss broader related works (Section 9) and provide concluding remarks (Section 10).

2 Deep Neural Networks and Hardware Accelerators

The flexibility and effectiveness of DNNs have made it the de facto solution for various applications, e.g., facial recognition [54]. Commonly used DNN architectures include, Multi-Layer Perception (MLP) [120], Convolution Neural

Networks (CNNs) [73], Generative Adversarial Networks (GANs) [42], diffusion models [47], and Transformers [123]. Lightweight models, for example, Binary Neural Networks (BNNs) [102] or compressed VGG-16 [109], can be deployed on edge devices without overly relying on cloud computation resources [43, 102].

For deployed DNNs, there are three main components of interest to adversaries: *the input*, *the architecture*, and *the trained parameters*. Depending on the application of the model, the input can take different forms, e.g., an image or a spoken sentence, for image and speech recognition, respectively. The architecture of DNNs defines the types and connections between the layers used in the network. Layers apply different transformations on their input data and forward their output to the subsequent layer(s). Each layer has a set of parameters, typically the *weights and biases*, that can be tuned so that the DNN can solve a certain task.

The architecture of a DNN consists of multiple layers, i.e., the building blocks that process and transform input data. The model part of Figure 1 provides an illustrative example of MLP and shows the calculation of one intermediate value in the first layer. While the various types of networks differ in architecture, the modules they use often overlap and share many similarities [45, 53, 58, 105]. Some common modules of these networks are used in different types of DNNs. By changing the variables in the configuration of these common modules, known as the *hyperparameters*, different DNN structures can be established.

Convolution layers and pooling layers are critical and fundamental building blocks in several types of Neural Networks, which are primarily employed for image recognition [72]. A convolution layer contains small *kernels* (also known as filters) that have trainable parameters and a predefined *shape*. During the forward pass, kernels convolve across the whole input with a certain step size, known as the *stride*. Pooling layers help control the number of parameters to control overfitting and reduce computational load.

Fully-connected layers (FC layers) have hidden nodes and each node applies a weighted sum over all inputs. The outputs of the layer are the weighted sums of each node, so the output size is determined by *the number of nodes* in the layer.

Activation functions introduce non-linearity into the DNN, enabling it to solve complex problems. Different types of activation functions are designed to increase efficiency [91] or reduce the vanishing gradient problem [44]. Typically used activation functions are *ReLU, Tanh, Softmax, and Sigmoid* [32].

Computational resource limit. Powerful DNNs require large computational resources, especially during training. Therefore, large DNNs are usually trained on GPUs, which by design greatly support parallel processing, to make this process more efficient. The inference latency of DNNs is critical for many applications, especially for resource-constrained environments such as embedded devices [74]. To this end, many accelerators have been developed, e.g., TPUs [65], and DPUs on FPGAs [8].

Implementations of DNNs on edge devices are further limited by available resources. To resolve this issue, resource-aware architectures have been designed, e.g., MobileNet [53] and EfficientNet [115]. Another solution is to quantize regular DNNs to reduce resource consumption during implementation, e.g., XILINX FPGA-based accelerators [8], and BNNs [102]. Hardware-wise, FPGAs are also used for edge applications where resources are limited. Due to their low computation overhead, low latency, and flexibility, FPGAs are popular choices for model inference [8, 112, 136].

3 Background on Side-Channel Analysis

Due to their physical implementation, electronic devices emit unintentional information about their state via *side channels*. Side-Channel Analysis (SCA) can be applied to obtain information about the executed operations and data processed by the device. Side channels can be either *physical* (e.g., timing, power, electromagnetic emanation) or *logical* (e.g., memory access patterns), based on the measured properties. In addition, SCA attacks are *non-invasive*, *passive* attacks where the adversary does not tamper with the normal execution of the device, just observes information, as opposed to fault attacks [13] which induce errors in the device's execution. In this work, we only consider passive physical SCA attacks that exploit timing, power or EM emanations as side channels.

Timing analysis exploits the variations in an algorithm's execution time where the amount of variation depends on the instruction executed or some other secret information. Variations in timing can occur for various reasons, e.g., non-constant time instruction, cache hit/miss, or branching [68]. In the context of neural networks, changing hyperparameters that alter the structure of a neural network can result in large variations in execution times [11].

Power analysis leverages the dependency of dynamic power consumption of an electronic device while processing and the data used in executed operations [69]. There are different flavors of power analysis of which we will focus on *Simple Power Analysis (SPA)* and *Differential Power Analysis (DPA)*. SPA analyses a few, or even just a single collected trace to extract information about the operations executed by the device or the used (secret) data. DPA and its variants, e.g., Correlation Power Analysis [16], recover secret data exploiting the data dependency of power consumption [83]. These dependencies can allow an adversary to perform statistical tests between the set of collected traces T and the hypothetical power consumption of secret data-dependent intermediate values of the algorithm, by using a leakage model, e.g., Hamming-weight (HW) or Hamming-distance (HD). In the context of neural networks, secret data are considered to be the parameters (i.e., the weights and biases) of the models in contrast to keys for crypto implementations.

Electromagnetic analysis takes advantage of electromagnetic (EM) emanations occurring due to moving charges that produce and change magnetic fields. Therefore, the EM emanations of a device executing an operation of interest can be measured by using an antenna to collect near- or far-field emanations from the target device. Since EM emanations depend on current flowing through components in a device, the concepts of SPA and DPA can also be used on EM emanations and they are called Simple EM Analysis (SEMA) and Correlation EM Analysis (CEMA), respectively. EM analysis has proven effective in eavesdropping on display units and breaking cryptographic implementations [36, 38, 50, 71, 80, 121].

Defenses against Side-Channel Analysis Thanks to the extensive research in the field, a broad range of different defenses can be used to protect devices against SCA. First of all, timing vulnerabilities can be mitigated by making implementations constant time with using, e.g., non-variable time instructions and only non-secret data-dependent branches. Defending against power analysis can be done both in hardware and software. In hardware, different logic styles than CMOS such as e.g. dual-rail [118] can be used for the hardware design (and production) of the device. In software, one can use masking, making the power dependent on other (masked) data, instead of actual data [24, 83, 101]. Finally, for EM analysis, specific hardware countermeasures, such as shielding, can prevent sensitive electromagnetic emanations leaking from the device [25].

4 Reverse Engineering Neural Networks

A trained DNN has multiple valuable facets that an adversary might try to acquire after the model is deployed:

1. the *inputs* during inference,
2. the *architecture* of the deployed model,
3. the *parameters* of the deployed model.

Inputs to DNNs can contain valuable or privacy-sensitive information, such as in medical applications [103]. The exact DNN architecture is IP and often kept secret because many cutting-edge models are closed-source and are usually modified versions of common architectures. For example, ChatGPT-4 [96] is a proprietary IP based on the Transformer architecture [123]. More importantly, the training of closed-source models can also include proprietary training data that an adversary may not be able to access. Furthermore, training a model requires a high level of expertise that the adversary may be lacking. Therefore, the parameters of the trained model are also of a paramount importance.

In this paper, we consider works whose objective is to recover one or more of these facets by exploiting *physical* side channels requiring proximity to the target device.

4.1 Threat Model

We decompose the threat model into *generic capabilities* and *objective-specific knowledge* to taxonomize previous SCA attacks. The generic capabilities are used across all investigated attacks, whereas the adversary may have additional knowledge, depending on the objective of the attack. These additional capabilities that an adversary may have are listed here as objective-specific knowledge.

Generic capabilities:

G1: Physical proximity to the hardware device, for which no countermeasures are implemented.

G2: Capability to collect and analyze power and EM traces.

G3: The attacker has access to a profiling device, identical to the device under attack.

Objective-specific knowledge:

Inputs:

I1: The input dimensions.

I2: The architecture of the DNN.

I3: The parameters used in the DNN.

Architecture:

A1: The input dimensions and values.

Parameters:

P1: The input dimensions and values.

P2: The architecture.

P3: The hardware design of the device under attack.

4.2 Taxonomy

We provide a taxonomy in Table 1 for all existing research works on reverse engineering DNNs via physical side channels to our best knowledge. Specifically, different SCA attacks are categorized by the objective and intermediate objectives of the attack, objective-specific knowledge of the adversary, the model type, the platform on which they are deployed, the SCA methods, and the attack path.

Objectives and dependencies. Objectives of adversaries are three-fold, namely input, architecture, and parameters, which are not always independent. The architecture of a DNN is the first secret that adversaries would like to extract, and hence the foundation to extract the parameters. As seen in Table 1, all parameter extraction methods need access to architectures either conducting architecture extraction or assuming that the architecture is known. Input recovery is less dependent on the other objectives, where only a two known approaches take parameters into account [82, 125].

Intermediate Objectives. The intermediate objective specifies the key intermediate results to achieve the final objective. For architecture extraction, the intermediate objective can be either component-level (layer and activation types, the number of layers (#layers) or number of neurons (#neurons), and kernel size) or model-level (candidate model ranking from known architectures). For parameter extraction and input recovery, the intermediate objective is often the ranking of the

Table 1: Taxonomy for reverse-engineering DL implementations with physical SCA.

Paper	Objective			Intermediate Objective	Specific Knowledge	Attack Scenario			
	Arch.	Params.	Input			Model Type	Platform	Analysis	Attack Path
Hu, et al. (2020) [56]	✓			Layer Type and #Layers	–	CNN	GPU	CP	2a
Takatoi, et al. (2020) [114]	✓			Activation Type	–	MLP	CPU	SPA, CP	1 2a
Xiang, et al. (2020) [127]	✓			Candidate Model Ranking	A1	CNN	CPU	CP	2a
Yu, et al. (2020) [135]	✓			Layer Type and #Layers	A1	BNN	FPGA	SPA, CP	1 2a
Chmielewski, et al. (2021) [22]	✓			Layer and Activation Types, #Neurons	A1	MLP	GPU	SPA, CP	1 2a
Maia, et al. (2021) [81]	✓			Layer Type and #Layers	A1	CNN	GPU	SPA, HO	1 2b
Wolf, et al. (2021) [126]	✓			Candidate Model Ranking	A1	CNN	CPU	CP	2a
Buzer (2022) [17]	✓			Candidate Model Ranking	A1	CNN	FPGA	SPA, CP	1 2a
Liang, et al. (2022) [76]	✓			Layer Type and #Layers	–	CNN	GPU	SPA	1 2c
Joud et al. (2023) [64]	✓			Layer Type and #Layers	–	MLP & CNN	CPU	SPA, CP	1 2a
Sharma et al. (2023) [107]	✓			Candidate Model Ranking	A1	CNN	FPGA	CP	2a
Horvath et al. (2024) [52]	✓			Candidate Model Ranking	–	CNN	GPU	SPA, CP	1 2a
Batina, et al. (2019) [11]	✓	✓		Layer and Activation Types, #Neurons, #Layers, Float-32 Ranking (7 Bits)	A1, P1	MLP	CPU	SPA, CP, DPA	1 2a 3
Regazzoni, et al. (2020) [104]	✓	✓		Layer Type, #Layers, Binary Ranking (1 Bit)	P1	BNN	FPGA	SPA, CP, DPA	1 2a 3
Yli-Mäyry, et al. (2021) [132]	✓	✓		Layer Type, #Layers, Kernel Size, Binary Ranking (1 Bit)	P1	BNN	FPGA	SPA, CP, DPA	1 2a 3
Gongye et al. (2023) [41]	✓	✓		Hardware Architecture, Layer Type, #Layers, Kernel Size, Integer Ranking (8 Bits)	P1	CNN	FPGA	SPA, CP, DPA	1 2a 3
Dubey, et al. (2020) [29]		✓		Binary Ranking (1 Bit)	P1, P2	BNN	FPGA	DPA	3
Joud, et al. (2022) [63]		✓		Float-32 Ranking (8 Bits)	P1, P2	MLP	CPU	DPA	3
Yoshida, et al. (2020) [133]		✓		Integer Ranking (8 Bits)	P1, P2, P3	MLP	FPGA	DPA	3
Yoshida, et al. (2021) [134]		✓		Integer Ranking (8 Bits)	P1, P2, P3	MLP	FPGA	DPA	3
Li, et al. (2022) [75]		✓		Integer Ranking (8 Bits)	P1, P2	MLP	FPGA	DPA	3
Horvath, et al. (2023) [51]		✓		Float-16 Ranking	P1, P2	CNN	GPU	DPA	3
Maji, et al. (2021) [82]		✓	✓	Float-32 Ranking (7 Bits), Binary Ranking (1 Bit)	I1, I2, I3, P1, P2	CNN & BNN	FPGA	DPA	3 4a
Wei, et al. (2018) [125]			✓	Image Silhouette, Integer Ranking (8 Bits)	I1, I2	CNN	FPGA	SPA, SA, DPA	4a 4b
Batina, et al.(2019) [12]			✓	Float-32 Ranking (7 Bits)	I1, I2, I3	MLP	CPU	DPA	4a
Dong, et al. (2019) [27]			✓	Image Silhouette	I1	MLP	CPU	SA	4b
Thu, et al.(2023) [116]			✓	Image Silhouette	I2	BNN	FPGA	SA	4b

target value, determined by e.g. applying DPA on the obtained power traces [16]. This ranking is distinguished by the precision of the parameters or pixels and the actual number of bits that were recovered are stated in brackets.

Platforms. Side-channel leakage is hardware-dependent: each platform processes DNNs differently, resulting in different leakage patterns. Microcontrollers, CPUs, FPGAs, and GPUs are the primary devices used to deploy DNNs for inference. Knowing the properties and specifications of the hardware platform, the adversary can decide on the side channel, i.e., EM- or power-based side channel, and collect the respective traces.

Analysis. When the adversary obtains the traces through the desired side channel, the SCA approach needs to be determined. For each of the different properties of a DNN, different analysis methods can be leveraged to analyze the obtained traces. Figure 2 shows the different flows of analysis methods that are currently used for extracting the architecture, parameters, and recovery of the input. From Figure 2 it can be seen that there is a finite number of combinations of attack paths that can be used to obtain a certain subset of objectives, of which the known combinations are shown in the attack path column of Table 1. For architecture extraction, we consider 1 SPA, 2a Common Patterns (CP), 2b Hyperparameter Optimization (HO), and 2c hyperparameter derivation. Sec-

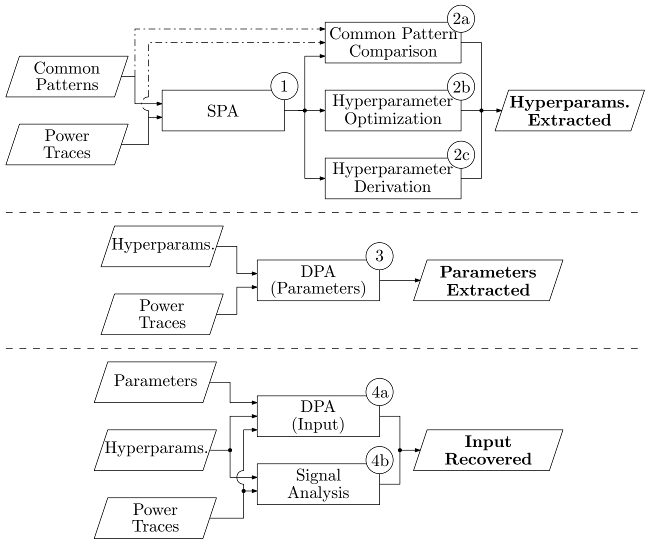

Figure 2: Physical SCA attack framework for model extraction and input recovery. Different attack stages are marked with different numbers, where a combination of different stages leads to an attack path. It should be noted that the three different targets require different inputs to the flow, where for both the parameters and input, the architecture should be provided.

ondly, for parameter extraction, we only consider $\textcircled{3}$ DPA. Lastly, for input recovery, we consider $\textcircled{4a}$ DPA and $\textcircled{4b}$ Signal Analysis (SA). The attack paths shown in Figure 2 can be combined to improve the threat model. For example, Gongye, et al. [41] use $\textcircled{2a}$ to extract the kernel size, similar to Xiang, et al. [127], which is then used to extract the parameters of the network. Detailed explanations for framework components can be found in corresponding sections.

5 Architecture Extraction

The first objective is to recover the architecture of the deployed model. The architecture of a model refers to the hyperparameters of a network, e.g., the number of layers in a network and their types. There are two main motivations for an adversary to extract the architecture of a network. First of all, the adversary may assume that the deployed model is a more general architecture and uses publicly available, pre-trained parameters. Therefore, knowing the architecture, the adversary could use pre-trained parameters to function as a substitute for the originally deployed model [5]. Secondly, the architecture itself could be a stepping stone towards the recovery of either the input or the trained parameters, as shown in Figure 2. For extraction of the architecture of a deployed DNN, a black-box scenario is being considered, i.e., the adversary has no knowledge of the deployed model, including the family of networks to which it belongs. The adversary only possesses the generic capabilities G1-G3 as described in Section 4.1 and may assume that the input is known (A1).

5.1 Current Approaches of Architecture Extraction

5.1.1 Simple Power Analysis

Some hyperparameters have shown to have very distinctive patterns for different values [11, 64]. These characteristics occur due to, e.g., the same operation having to be executed several times, or operations having varying execution times. The adversary can already extract information from the obtained traces by using SPA. Moreover, for most approaches towards architecture extraction, listed in Table 1, SPA is required as a first step, as shown in Figure 2. For example, Yu et al. first use SPA to recover the layer types, and then apply this knowledge to recover the remaining hyperparameters [135]. On the other hand, Xiang et al. [127] use a classifier trained on common configurations to extract information of the hyperparameters directly from the power trace, and hence do not use SPA. Due to the small resource footprint and proven effectiveness, SPA is in most cases preferred over more computationally-heavy approaches to extract hyperparameters. However, on more developed platforms, traces typically are less distinguishable, making SPA impractical.

Different types of layers in a DNN all consume power in a distinctive way, which is observed as variations in the power traces [41, 64, 76, 81, 135]. These patterns originate from the varying way in which the different types of layers access data [76]. Due to the differences in the power traces, the adversary is able to determine the type of layer that was executed, based solely on the observation of the recorded traces.
The number of layers can be determined from simply counting the different types of layers, in case of less powerful CPUs [81]. Due to the distinctive patterns, adjacent layers can be identified from the raw traces. These patterns can then be counted to determine the number of layers in the DNN. It should be noted that due to high parallelization in FPGAs and GPUs, simply counting the layers is often not possible.
Activation functions have a high variation in execution time, depending on their input. This varying execution time already makes it possible for an adversary to distinguish between the ReLU and Softmax activation function. However, for the Sigmoid and Tanh activation functions, the execution times are similar and not easily distinguished [11]. Nevertheless, the characteristic differences in power consumption of the Sigmoid and Tanh activation functions, allow them to be distinguished after all [114]. Therefore, the four most common activation functions, ReLU, Softmax, Sigmoid, and Tanh, can be often distinguished using SPA only.
The number of neurons in MLPs can also be determined from SPA, by observing the execution time. The execution time of each layer is dependent on the number of neurons in the layers of an MLP. From observing the execution time of MLPs, one can determine number of neurons in the network [11]. However, the measurement noise and re-alignment make it difficult to distinguish a single perceptron change [22].

5.1.2 Advanced Analysis

In practice, deployed models are part of a limited set of *common DNN models* [52, 76, 127, 135]. Therefore, the likelihood that a deployed model adopts a common architecture is higher than a custom-made architecture. Moreover, upon recovery of a limited number of hyperparameters, the remaining hyperparameters can be determined by applying the knowledge of the most common configurations for DNNs, depicted as the hyperparameter derivation step $\textcircled{2c}$ in Figure 2 [41, 56, 64, 81]. However, direct derivation of the remaining hyperparameters may not always be possible due to the lack of equations with unique solutions. To resolve this, an adversary can set fixed values for some of the unknown hyperparameters or use more advanced techniques to acquire these hyperparameters via physical SCA.
Common patterns are created to take into account the design philosophy of commonly used DNN models, from which assumptions can be made about the hyperparameter values. First of all, there is a common order in which layers can occur throughout a network, i.e., the type of layer $i+1$ can be de-

termined, provided the type of layer *i* by using knowledge of default layer combinations [56]. Secondly, hyperparameters such as the kernel size can be approximated when the layer type is known [76, 135]. For example, when considering a convolution layer, the used filter sizes in the most common models are either 1×1, 3×3, 5×5, or 7×7 [135]. For square kernels in BNNs, one could additionally observe the size of the kernel by executing a T-test on the input to distinguish between different filter sizes [104,132]. Additionally, a classifier can be made, based on the common patterns, to determine directly from the power traces to which of the standard hyperparameter combinations the trace belongs [17,52,107,126,127]. Therefore, the correct hyperparameters can be extracted from the observed power traces, given common patterns.
Hyperparameter optimization can also help to fine-tune an estimate from commonly used hyperparameters. Typically, an initial estimation based on commonly used patterns of hyperparameters is not very accurate. It could be that, e.g., a wrong value for the stride has been chosen which causes the derived model to deviate from the original target model. Therefore, Maia et al., decided to jointly optimize all hyperparameters "*seeking values that best fit their initial estimates, subject to consistency constraints*" [81]. The technique of Maia et al., minimizes the convex quadratic form

$$\min_{x_i\in\mathbb{Z}^{0+}} \sum_{i\in X} (x_i - x_i^*)^2, \qquad (1)$$

where X is the set of all hyperparameters, and x_i^* and x_i are the initial and optimal value for the i^{th} hyperparameter [81]. Hence, optimization techniques could be used to extract a more accurate approximation of the target architecture [81].

5.1.3 Assumptions on Architecture Extraction

For all of the architecture recovery methods in Table 1, it is assumed that target models are common DNN configurations. Moreover, assumptions for single hyperparameters can also have consequences for other hyperparameters, e.g., restricting the output size of a layer inherently restricts the input size of the next layer [41]. Therefore, putting restrictions on architecture configurations reduces the search space for the hyperparameters, making it easier to recover the architecture of the target DNN.
Only four activation functions are assumed to be used in common DNNs, i.e., only ReLU, Sigmoid, Tanh, and Softmax are typically considered for activation functions [11, 22, 64, 114]. In theory, there exist more activation functions than only these four, e.g., ELU or the binary step function [23, 32]. In practice, because of the popularity, it is a valid assumption that the activation function used in a DNN is ReLU, Sigmoid, Tanh, or Softmax [32].
Only limited layer types are considered when extracting the hyperparameters of a DNN. For BNNs, the only two layer types that are used, are the pooling and convolution layers, which also reduces the search space of the hyperparameters [104, 132]. Additionally, the combination of layer types can be reduced based on common DNN implementations. For instance, it is unlikely that a convolution layer is followed by an FC layer, since it does not make sense to have two consecutive linear transformations in a DNN [56]. Additionally, the combination of layer types can be reduced based on common DNN implementations [56]. For instance, it is unlikely that a convolution layer is followed by an FC layer, since it does not make sense to have two consecutive linear transformations in a DNN. Hence, assumptions made about the layer types can also simplify the extraction of the entire network architecture.
The kernel is considered a square in all of the approaches listed in Table 1, that aim to extract the architecture. Since almost all publicly deployed models, in fact, use a square kernel, this assumption is valid. Nevertheless, there exist configurations in which the kernel is not a square [113], which are not considered in the discussed architecture extraction papers. Moreover, most papers listed in Table 1, limit the size of the filter to typically odd dimensions between 1 and 11. Due to the limited set of filter dimensions that is currently being deployed, it is reasonable to assume for an adversary that the filter dimension is also part of this limited set.
The stride is assumed to be 1 or 2 in most papers attempting to extract the architecture [41, 56, 76, 81]. Due to the limited size of the kernels that are being considered, a bigger stride could result in skipping input values during the convolutions, or the model not being feasible at all. Therefore, an increment of the stride on a DNN that has a small filter size could result in a lower accuracy of the overall model. Moreover, the most common DNN implementations also rarely use a stride that is bigger than 2 [76]. Hence, assuming the stride is limited to either 1 or 2 is a valid assumption that further reduces the search space for hyperparameters.

Aside from the most common assumptions listed above, there are additional assumptions that are still worth mentioning. One obvious assumption is that for image classification applications there are only three input channels used in the first layer [56]. With most input images being of color, all of them have red, green, and blue (RGB) channels representing the pixels. Aside from RGB images one could also consider only grayscale images, in which only one channel is considered. Secondly, in some scenarios it can be assumed that for the same layers in the network, the same configuration is being used, i.e., the first convolutional layer uses the same filter size as all other convolutional layers in the network [135].

5.2 Discussion on Architecture Extraction

Within the assumption spaces of each work, the full architectures of deployed DNNs can be extracted. Most of the works listed in Table 1 have a number of assumptions, hence considering only a subset of all the possible configurations for the architecture. We argue that these assumptions are, in most cases, correct and can help simplify the attack process, but may fall short in less common but nevertheless, appropriate use cases. In particular, this section provides experimental analysis on the well-acknowledged assumption that the kernel shape in convolutional layers is always considered to be a square. Even though the square-shaped kernel is efficient to implement and is heavily used for image filtering in computer vision research, non-square convolution kernels are still being widely used [113].

If a deployed model has a non-square shaped kernel but the adversary classifies it as being a square, the recovery of the architecture will fail. Consequently, further extraction of the model may fail as well and result in a non-functioning model. In Figure 3, traces related to a 224×224 input convolved with a 1×4 and a 2×2 kernel are collected on a Xilinx Zynq UltraScale+ ZCU 104 board [9]. As can be seen from Figure 3, the traces for both square and non-square kernel are very similar; it is hard to distinguish them at first glance. If the adversary assumes that the kernel is of a square shape, and observes a trace for a non-square kernel, as shown in Figure 3, it may result in the adversary incorrectly classifying the kernel as a square. Therefore, if a limited set of configurations is considered, but the deployed model is in fact a corner-case it may result in an unsuccessful recovery of the architecture.

However, if one closely observes the traces in Figure 3 it can be seen that there is in fact a slight timing difference between the two traces. This timing difference between the traces can be used for distinguishing between the two different kernels. Therefore, if the adversary has the capacity obtain a pattern for multiple kernel configurations, it would be possible to distinguish between different kernel configurations.

Note that the traces in Figure 3 represent the execution time and the power consumption during the convolution. The power consumption and execution time of the observed traces are related to the number of operations that are being executed [127, 135]. For convolution layers, these operations are mainly the number of multiplications and loading of the values required for multiplication. The latter is dependent on the number of convolutions that are being executed. Therefore, the shape of the traces is not only dependent on the number of multiplications, but also on the number of convolutions. Given a kernel of height k_h and width k_w and an input of height i_h and width i_w, the number of convolutions C can be described by

$$C = \frac{i_w - k_w + 1}{s} \cdot \frac{i_h - k_h + 1}{s}, \tag{2}$$

where s is the stride. Consequently, the total number of multiplications M in a convolution layer can be described by

$$M = k_h \cdot k_w \cdot C. \tag{3}$$

It can be seen from eq. (2) and (3) that $\{C, M\}$ is not unique, i.e., there are different values for k_h, k_w, and s that will result in the same $\{C, M\}$. Therefore, theoretically different kernel configurations could result in similar power traces.

An example of different configurations for which the same number of convolutions and multiplications are used is when a kernel has a size of $k_1 = (k_{h1}, k_{w1})$ and the second kernel has a size of $k_2 = (k_{w1}, k_{h1})$. Figure 4 shows the power traces of a convolution layer with a 28×28 input convolved with both a 3×4 and 4×3 kernel. For both kernel configurations in Figure 4 it holds that both the number of convolutions and the number of multiplications are equal. However, Figure 4 shows that even with same number of multiplications and convolutions, the observed EM traces allow the adversary to distinguish between the two configurations.

Since equivalent configurations in terms of convolutions and multiplications can be distinguished through power traces, including additional configurations allows for the architecture extraction method to extract a bigger set of possible architectures. This expansion of the search space of an architecture extraction approach improves its applicability to different, possibly unknown, architectures. Therefore, relaxing assumptions about deployed architectures can help approaches introduced by, e.g., Hu, et al. [56] or Maia, et al. [81], to extract architectures that deviate from common configurations.

5.3 Conclusion on Architecture Extraction

Current approaches to architecture extraction extensively use hyperparameter-search-space limiting assumptions. Even though these assumptions are well grounded, it may be the case that the deployed model lies beyond the defined search space for hyperparameters. Due to high similarity between traces, it may be the case that the adversary incorrectly determines hyperparameters, resulting in an inaccurate architecture being extracted. However, we demonstrate that investing in expansion of the search space for hyperparameters could decrease this misclassification.

6 Parameter Extraction

With the trained parameters of the network, an adversary has all the information necessary to deploy an exact copy of the network. This could give the adversary both monetary and intellectual benefits. For instance, when an adversary possesses the trained parameters of ChatGPT-4 [96], he or she does not need to pay the subscription fee and can let others use the extracted model. Therefore, the trained parameters represent the most valuable part of a DNN, and hence an interesting target for an adversary.

In addition to the generic capabilities G1-G3 described in Section 4.1, the adversary has more knowledge about the deployed network when extracting the trained parameters. In particular, for parameter-recovery, the adversary knows the input to the model (P1), and knows the architecture of the deployed DNN (P2). For P1, it is a standard assumption in

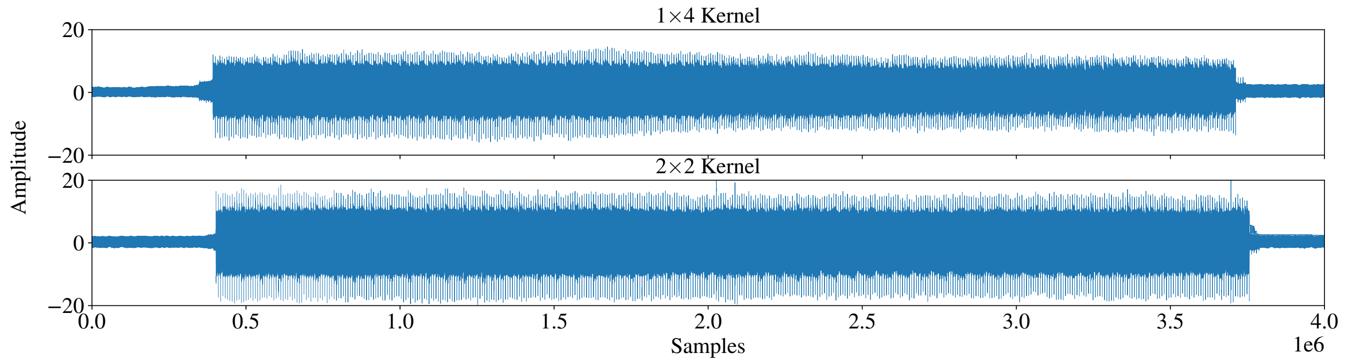

Figure 3: Averaged EM traces of convolutional layers with different 2D kernel sizes, taking 224×224 grayscale images as inputs. The top trace shows the execution of a convolutional layer with a 1×4 kernel, while the bottom trace uses a 2×2 kernel.

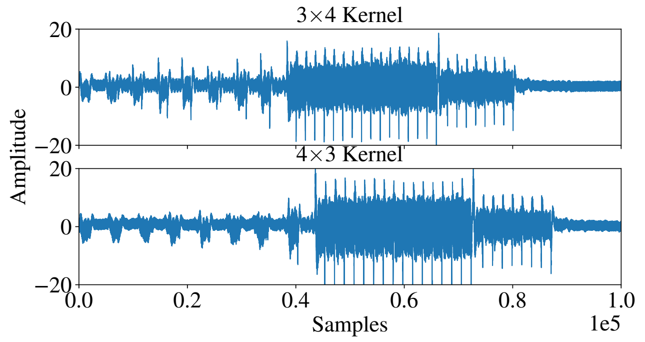

Figure 4: Averaged EM traces of convolutional layers with different 2D kernel sizes, taking 28×28 grayscale images as inputs. The top trace shows the execution of a convolutional layer with a 3×4 kernel, while the bottom trace uses a 4×3 kernel.

physical SCA attacks that the adversary knows the inputs [83]. For P2, we show that extraction of the architecture is possible with physical SCA approaches.

6.1 Current Approaches of Parameter Extraction

6.1.1 Differential Power Analysis

To extract the parameters of FC and convolutional layers, all current approaches towards parameter extraction in Table 1 use DPA to correlate measurements with the power consumption of hypothetical sensitive intermediate values, indicated by ③ in Figure 2. Since FC and convolutional layers are usually implemented as matrix multiplication operations [21], an adversary can target the multiplications and partial sums to extract the parameters.

In a 2D convolutional layer, with kernel size $k_h \times k_w$, the convolution result c_{res} between a kernel (w) and a piece of input x can be unrolled like

$$c_{res} = \mathbf{x} * \mathbf{w} = \sum_{i=1}^{k_h \cdot k_w} w_i \cdot x_i = w_1 \cdot x_1 + \cdots + w_{k_h \cdot k_w} \cdot x_{k_h \cdot k_w}. \quad (4)$$

The sensitive intermediate values in the unrolled convolution are the partial sums $c_j = \sum_{i=1}^{j} w_i \cdot x_i$ $(j = 1..k_h \cdot k_w)$ that are updated with the result of the multiplication of an input x_i and weight w_i.

Since the inputs are known, the attacker can compute the Pearson correlation between the traces (T), and the leakage model (L) of the i^{th} intermediate value $\rho(T, L(c_i))$ for every parameter candidate, where the candidate parameter for which $\rho(T, L(c_i))$ is the highest is considered to be the correct value for the targeted parameter [11, 29, 41, 51, 63, 75, 82, 133, 134]. Typically, due to the inter-dependency of the parameters, shown in eq. (4), the DPA approach recovers the weights one-by-one starting from w_1 until $w_{k_h \cdot k_w}$. However, convolution is a linear operation where multiple weight candidates can give the same hypothetical power consumption values which makes the DPA attack harder. Therefore, applying the HD leakage model, to model the behavior when a partial sum is updated can give better results [41, 51].

6.1.2 Developments of Parameter Extraction

All parameter extraction approaches in Table 1 leverage DPA, however the approaches can still be distinguished from each other based on the precision of the parameters and the performance of the approach. An example can be seen in Yoshida, et al. [133], Yoshida, et al. [134], and Li, et al. [75], where all of them leverage DPA to recover 8-bit integer parameters from MLPs deployed on FPGAs. Yoshida, et al., Yoshida, et al., and Li, et al. all face the same issues whilst extracting parameters through DPA: the parameters are inter-dependent, as shown in eq. (4). Each of these approaches cope with this inter-dependency in their own way, always improving on the latest research.

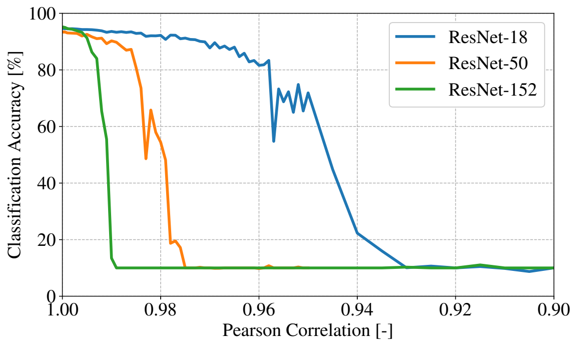

Figure 5: The classification accuracy of ResNet-18, ResNet-50, and ResNet-152 trained on CIFAR-10, plotted against the Pearson correlation between the original and transformed parameters. With the same architecture, models with more layers are more sensitive to measurement inaccuracy.

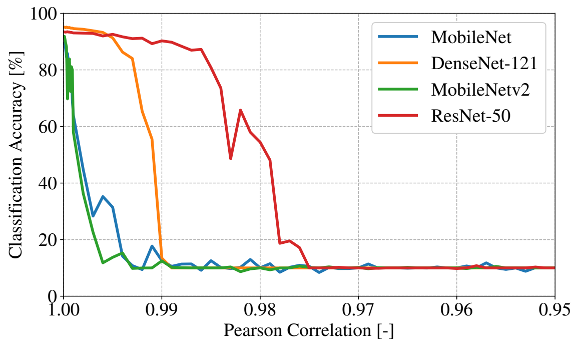

Figure 6: The classification accuracy of MobileNet, DenseNet-121, MobileNetv2, and ResNet-50, trained on CIFAR-10, plotted against the Pearson correlation between the original and transformed parameters. Here, sampled noises from Gaussian distribution are added to the kernels to simulate hardware-based environmental noise.

6.1.3 Assumptions on Parameter Extraction

In order to extract the parameters, current approaches need to access the exact inputs and the architecture of the deployed model [11, 29, 51, 63, 75, 82, 132–134]. Otherwise, accurate calculation of sensitive intermediate values is not possible, and the DPA attack will not be successful. In addition, the search space for the parameters is often limited to a range with certain precision [11, 51, 63, 82]. This allows an adversary to significantly reduce the number of candidates in DPA attack, especially if the parameters in the models use single-precision or double-precision data format. Furthermore, some attacks require that the inputs to the DNNs are carefully chosen but this assumes a stronger attacker that is able to control the inputs [41]. In addition, even if the victim's model parameters are updated after extraction, the adversary can still retrieve a functioning model that can be further trained with their data.

6.2 Discussion on Parameter Extraction

The DPA method described in Section 6.1 is a general solution that works across platforms and different data types to extract the parameters of a DNN. For BNNs, the accuracy of the extracted parameters through DPA even goes to 100% for the best method [132], and 99.99% for other methods [29]. However, as the data format of the parameters includes more bits, the more difficult it becomes to extract the correct parameters, and there can be errors in the recovered parameters [11].

There are two major limitations related to the state-of-the-art extraction of the parameters in DNNs: *parameter-extraction error propagation* and *complexity*. Firstly, it is yet unclear how the inaccurate extraction of parameters in one layer impacts the extraction of parameters in subsequent layers. Currently, none of the papers address this issue for large DNNs. Secondly, extracting all the parameters in DNNs is expensive. All papers note that the extraction of the parameters is an expensive operation, however the consequences for the extraction method are only slightly touched upon and notions of complexity are often missing. In this section, we will elaborate on the limitations present in the current extraction methods of DNN parameters.

6.2.1 Error Propagation of Parameter Extraction

Except for SCA attacks on BNNs [104, 132, 135], most previous works focus on extracting the parameters of the first layer in DNNs. Previous works hypothesize that it is feasible to reverse engineer entire DNNs by showing the successful extraction of the parameters of the first layer with a low error rate [11]. We seek to evaluate this hypothesis critically by detailed analysis and simulations in comparable scenarios.

Analysis on error propagation. We first provide an analysis showing that the error in parameter extraction can be propagated from shallow layers to deep layers in DNNs, and the error can accumulate layer by layer. In DPA, to estimate the parameters of a DNN, the adversary first needs to select an intermediate value and collect the corresponding traces. Given the search space of possible candidate values and the quality of collected traces, it could be that a large number of traces is required to predict the correct parameter. Even though a massive number of traces can be collected e.g., tens of millions, there is still a probability that predicted intermediate values deviate from the actual values due to noisy measurements, resulting in inaccurate parameter predictions. Since the intermediate values of the deeper layers of DNNs are calculated based on the estimated parameters of the previous layers, the error in deeper layers is amplified by inaccurate intermediate values in shallower layers. This inaccuracy of the measured traces is mainly due to noise induced in the measurements originating from environmental sources and is considered to be stationary, uncorrelated, and added to the power traces.

Therefore, this noise can be treated as a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ [92]. Take, e.g., MobileNet [53] for which the average extraction error on the first-layer feature map (i.e., intermediate values) is 0 with a variance σ, and assume that the error follows a Gaussian distribution. Accordingly, the same error applies to predictions of the first-layer convolution kernels. Each value in the second-layer feature map depends on the 27 convolution kernels from the first layer if we know the activation and pooling functions. The accumulated error propagated from the previous layers then becomes approximately $\mathcal{N}(0, 27 \cdot \sigma^2)$. Hence, error propagation will be more influential in deeper DNNs, as shown in Figure 5 for ResNet-18, ResNet-50, and ResNet-152, with 18, 50, and 152 layers, respectively.

Simulation on error propagation. We provide a simulation to model the influence of the error propagation on the performance of the extracted model. We conduct experiments by simulating extracted parameters from MobileNet [53], MobileNetv2 [105], DenseNet-121 [58], and ResNet-50 [45] trained on CIFAR-10 [70], and we use classification accuracy on test data to represent the performance of the extracted model. In particular, we sample noises from Gaussian distributions which we add to the original parameters, to simulate errors in the extracted parameters. We then describe the introduced error by calculating the Pearson correlation between the original and extracted parameters. Figure 6 demonstrates that a high correlation may not guarantee the performance of an image classifier. From Figure 6 it can be seen that, e.g., MobileNet requires the correlation to be higher than 0.9994 to keep the classification accuracy above 77.80%. This required 99.94% extraction accuracy is above the 99.75% extraction accuracy that can currently be achieved, making it infeasible to accurately recover the parameters of MobileNet with the current approaches listed in Table 1.

Figure 6 shows that the architecture determines the sensitivity of the network against inaccurate measurements. If we compare, e.g., DenseNet-121 with 121 layers, and MobileNet with 27 layers, MobileNet requires an extraction accuracy of 99.95% to stay above a classification accuracy of 80% whereas DenseNet-121 only requires an extraction accuracy of only 99.7%. This is counter-intuitive, since we argued that more layers typically mean more error propagation and thus, typically, a lower performance of the extracted model, as demonstrated in Figure 5 for ResNet. Therefore, not only the number of layers, but also the overall architecture of the network determines the sensitivity against inaccurate measurements.

6.2.2 Complexity of Parameter Extraction

All methods for parameter extraction listed in Table 1, assume that the approaches can be used in deeper DNNs. Most of the parameter extraction approaches listed in Table 1 demonstrate the feasibility of their approach by only considering a single 3×3 kernel, i.e., recovering 9 weights. However, these approaches fail to mention how the solution can be applied to state-of-the-art DNN architectures, which can have millions of parameters. Therefore, the complexity of these approaches needs to be addressed.

We address the complexity of the DPA approach towards parameter extraction both from a measurement and an analysis point of view. First of all, during measurements, each collected trace contains d samples that should involve only the operations of interest, i.e., the operations where the parameters are involved. The larger the DNNs that are considered, the more samples it takes to fully capture their execution in a trace. Consider that in a convolution layer with a kernel size $k = k_h \cdot k_w$ there are C convolutions of k multiplications, then for a kernel of size $k+n$ there are $C \cdot n$ additional multiplications. These $C \cdot n$ additional multiplications then increase the number of samples that reside in a single trace with $n \cdot d/k$. Therefore, trace collection has a complexity of $O(n)$.

Secondly, for the analysis part of the DPA approach the obtained traces are correlated with all possible candidates for the parameters, of which the candidate with the highest correlation is being selected. The values for the candidate parameters of non-binary DNNs generally reside in a range $[-L/2, L/2]$, with $L \in \mathbb{N}$, and a precision p, so there are $N = L/p$ different candidates for each weight. Therefore, if we consider a kernel with n different weights, theoretically, there are N^n different combinations of candidates for the entire kernel. Nevertheless, extracting the parameters from the partial sums as described in Section 6.1 has a complexity of $O(N \cdot n)$. Moreover, with fixed kernel size, the complexity is linear w.r.t. the number of weights, i.e., $O(n)$. Therefore, the complexity of current approaches towards parameter extraction linear for both the trace collection and the analysis parts of the DPA approach.

6.3 Conclusion on Parameter Extraction

In conclusion, there are two fundamental issues with the current approach to use DPA as a method towards parameter extraction of DNNs. First of all, we showed that even the slightest errors in the recovered parameters can result in a non-functional model. Secondly, due to the linear complexity of recovering the parameters, the DPA approach can become computationally expensive for models with millions of parameters. Therefore, there exists a gap between the theoretical feasibility and practical implementation of current DPA approaches towards extracting the parameters in DNNs.

7 Input Recovery

In this section, we look at input recovery from a deployed DNN during inference. State-of-the-art CCTV cameras are equipped with a DNN chip that allows for facial recognition, which makes the input to the deployed DNN privacy-sensitive information [15]. Even with physical access to the target device, the inputs might be encrypted in transit and only decrypted during inference. Therefore, methods such as monitoring I/O ports, network traffic, wireless communications, may not be able to recover the inputs. Instead, an adversary could launch an attack against DNN implementations to obtain the input, making it an interesting target for input recovery.

For recovery of the inputs, comparable to extraction of the parameters, the adversary has additional capabilities than the generic capabilities G1-G3. In both input recovery methods 4a and 4b shown in Figure 2 the adversary has different capabilities. For the DPA approach 4a, it is assumed that the adversary knows the architecture (I2) and the parameters of the deployed DNN (I3). For the signal analysis approach 4b, it is assumed that the adversary knows the architecture (I2) but not the parameters of the DNN.

7.1 Current Approaches of Input Recovery

7.1.1 Power Analysis

As can be seen from eq. (4) in Section 6.1, the intermediate values are not only dependent on the parameters, but also the input values. Then, for the first multiplication step ($i = 1$), the intermediate value in the multiplication is the partial sum which depends on both the input and weight. Therefore, recovery of the input can be already done in the first layer, in a similar manner as extraction of the parameters.

Similar to DPA for parameters, the possible input candidates are being correlated to the observed traces. In contrast to parameter extraction, the input is typically only used exactly once, e.g., an input frame of a CCTV camera will never be exactly the same as the previous one in a natural environment. Therefore, the adversary only has one shot to obtain the traces for the input recovery, for which correlating the entire trace to the candidates will not be sufficient to obtain a significant correlation difference between correct and incorrect candidates [12, 82, 125]. To cope with this limited number of available traces, knowledge about the architecture (I2) is leveraged to divide the traces in the corresponding multiplication operations. The known parameters are used to create hypotheses about possible intermediate values. These hypotheses are then correlated against the traces of singular multiplication operations to approximate the original values of the input [12, 82]. Additionally, when there is no knowledge about the parameters, but instead it is possible to provide input to the target model in a profiling stage (I1), a "*power template*" can be made [125]. This power template will be built by observing power trace from the target model with known inputs. In this way, the observed power traces from the target model, with the target input, can be matched to the generated power template to obtain a candidate for the input.

7.1.2 Signal Analysis

Even though it recovers exact values for the input, DPA is very expensive and it may not always be necessary to fully recover the input. The silhouette of an image may in some cases already disclose sufficient information for an adversary [125]. If the complete input image is not required, less computational-intensive methods can be used.

SA is an extension to SPA, where instead of directly interpreting patterns in the traces, additional computations have to be performed to extract information, e.g., interpreting bits from obtained traces [116]. As with DPA, SA approaches also exploit the data dependency in the power consumption of the deployed device, but without any knowledge of the parameters. Unlike DPA, the acquired traces can be processed individually rather than computing the correlation with other traces. Since the multiplications are dependent on the intermediate values, and hence the inputs, observing the magnitude and timing of the traces can already reveal information about the input, e.g., higher amplitudes or longer operations in the traces imply more computational intensity and hence it is most likely higher input values were used [27, 116, 125]. Similar to DPA, knowledge about the architecture (I2) can be used to split the observed traces in the different multiplication operations, making SA more accurate [125].

7.1.3 Assumptions on Input Recovery

Since inputs are recovered by observing similar leakages as for parameters, the assumptions made for input recovery are alike. Firstly, for recovering the inputs via DPA two assumptions are commonly used: the assumption that the architecture is known (I2) and the assumption that the parameters are known (I3). The architecture of the network reveals how the multiplications are executed and gives insight in how the traces are built up. To improve the recovery process, it is helpful to not only know when the multiplication operations are executed, but also what exact values are used. Therefore, knowing the parameters of the network could reveal more details about the inputs of the network, hence improving the recovery process [12, 125]. Secondly, for SA the only assumption made is knowledge about the architecture. This knowledge about the architecture is used, like for DPA, to split up the traces into single multiplication operations.

7.2 Discussion on Input Recovery

The state-of-the-art physical-SCA-based input recovery approaches have proven to be successful with single-shot traces [12, 116, 125]. For SA approaches, there is no strict requirement on the knowledge of the network, but here only attribute inversion can be performed, i.e., only partial information can be extracted about the inputs. On the other hand, for DPA approaches the adversary needs to know both the architecture and the parameters of the network to make ac-

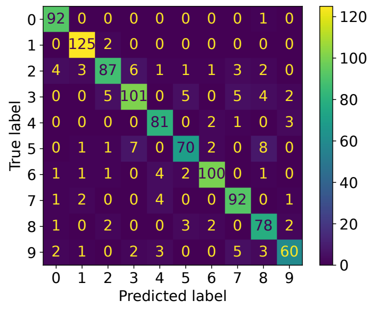

Figure 7: Prediction confusion matrix for inference-trace-based input prediction on 10 digits.

curate hypotheses. These strict requirements allow full input recovery to be possible.

However, as discussed in Section 6.2, if the adversary extracts inaccurate parameters, this inaccuracy may propagate through subsequent layers. Even though only one layer has to be considered, this error still propagates, through the DPA hypotheses, to recovered inputs. Therefore, if the adversary deals with a black-box scenario and wants to perform DPA, the adversary relies on the accuracy of extracted parameters.

To overcome the dependency on parameter extraction accuracy, we explore a new attack angle towards input recovery. In particular, we relax the purpose of the attack from reconstructing the raw input. Instead, we focus on attribute inversion, i.e. inferring sensitive attributes of the input such as the class it belongs to. This is similar to the "*passive adversary*" introduced by Wei et al. [125]. However, unlike Wei et al., we weaken the adversary and assume that, in addition to the exact parameters, also the architecture is unknown. We conduct a digit classification task on the MNIST [72] test split, for which the inference is implemented on a Xilinx Zynq UltraScale+ ZCU 104 board. EM traces are collected and then annotated during the digit inference, where 90% of the acquired traces are used to train and validate an X-vectors model [110]. Input digits can be predicted with an accuracy of 88.60% by EM traces during model inference, with details shown in the confusion matrix in Figure 7. Note that the number of different MNIST digits is not uniform, and our results are provided on the last 10% of the test split. Additionally, we attempt to distinguish the traces for different digits using SPA. However, the traces are indistinguishable from each other when using only SPA.

From the proof-of-concept discussed above, it can be seen that without any knowledge of the architecture, still attribute inversion can be performed on the input of the network. This opens up a new avenue of input recovery attacks, where attributes of the input can be recovered through physical side-channel attacks. Therefore, input recovery attacks may pose a larger threat than considered today.

Table 2: Physical SCA as an intermediate step for adversarial machine learning attacks. ●: Mounting existing adversarial machine learning attacks can directly exploit or primarily benefit from the extracted DNN information. ◐: Potential to benefit adversarial machine learning attacks. ○: Uncertain yet whether the extracted DNN information could benefit.

Objective	Evasion	Poisoning	Membership Inference	Model Inversion	Model Stealing
Arch.	●	◐	○	○	●
Params.	●	●	●	●	●
Input	◐	◐	○	○	◐

7.3 Conclusion on Input Recovery

The dependency on precise model parameters makes input recovery a less practical physical SCA. By adjusting the threat model and relaxing the dependency on model parameters and architecture, we demonstrate that input recovery could still be effective without deviating from the attack objective.

8 Outlook

SCA assumptions with substantial limitations. Current research makes strong assumptions about the knowledge and capabilities of the adversary. These assumptions are rarely subjected to comprehensive critical examination. We introduce a generic threat model and systematically classify all related existing research within this generalized framework. Furthermore, we encourage future studies to provide discussions on the constraints inherent to these assumptions.

SCA for novel neural network architectures. Regular SCA attack scenarios assume that all the components of DNNs are available for profiling, i.e., it is assumed that all layer types are publicly known. When a new type of module is introduced in the architecture and the component information is unavailable due to IP, profiling may fall short. As a consequence, the model extraction or input recovery attacks may fail. Moreover, current neural network extraction research is focused solely on MLPs, CNNs, and BNNs, while the interest for GANs, diffusion models, and transformers is increasing. Future research could look at the extraction of previously unexplored DNNs by either decomposing them into components [56] to extract or categorizing them in model type level [52, 127].

DNNs of FP8 formats The usage of the 8-bit floating point (FP8) binary interchange format in deep learning training and inference has been advanced by NVIDIA, Arm, and Intel [87]. FP8 in deep learning indicates a smaller parameter range and, accordingly, a smaller search space for SCA attacks. Such a smaller search space can substantially increase the SCA efficiency and reduce the error propagation (See Section 6). Future studies are encouraged to evaluate and provide simulation attacks against FP8 platforms.

Physical SCA as an intermediate step Physical SCA attacks not only pose a threat on their own but also have the potential to provide information to assist algorithm-level adversarial machine learning attacks. From all works discussed Table 1, only Maia, et al. provided a discussion on the use of their approach as an intermediate step for evasion [81]. To facilitate future research, we, for the first time, systematically bridge physical SCA attacks and well-established adversarial machine attacks in Table 2.

Architecture extraction provides a solid initial step for various adversarial machine learning attacks that target DNNs. Extracted architectures can assist the adversary in building surrogate models that help query-based [98] and transfer-based [26, 81] evasion attacks. Furthermore, details of the architecture aid the model stealing attack to extract the decision boundaries [61], and there is a potential to aid poisoning.

Parameter extraction extracts a white-box DNN that benefits all attacks from Table 2. Membership inference [108] and model inversion [37] attacks can directly exploit extracted parameters to compromise data privacy. Even with errors in the extraction of parameters as discussed in Section 6.2.1, parameter extraction can still assist different attacks. For evasion, extracted parameters with an error or less precision may not substantially compromise the attack performance [130].

Input recovery directly undermines inference data privacy. When more input samples are extracted, data distributions of the inputs can be estimated, which could help evasion and poisoning attacks. Furthermore, input recovery has the potential to optimize the sampling process when querying a black-box learning model, improving the efficiency for both evasion and model stealing attacks [61, 97]. To better connect the physical SCA extraction and existing threats of machine learning systems, we refer security researchers to the general threat matrix of machine learning systems [1, 2, 6].

Countermeasures In parallel to evolving attacks, countermeasures are being developed to protect against DNN extraction attacks. These countermeasures are similar to those protecting cryptographic implementations. Software-based countermeasures include, e.g., shuffling [93] and desynchronization [14] to protect against parameter extraction. On the hardware side, masking countermeasure is proposed to protect dense layers and the ReLU activation functions in BNNs [29]. Subsequently, boolean masking is proposed to protect BNNs against parameter extraction [28, 30]. Although effective, these countermeasures come at a speed and silicon area cost.

9 Related Work

Non-physical SCA. This SoK only discusses physical SCA approaches towards model extraction and input recovery, that measure physical quantities without inferring data directly from it. Aside from this scope, there exist SCA attacks that do not rely on physical side-channel measurements. First, the power consumption can be measured through other channels, e.g., internal monitors [31, 39, 85, 86, 89, 90, 95, 99, 117, 137]. Second, DNN properties can be extracted by observing the memory access patterns, for which methods such as Prime+Probe and Flush+Reload can be leveraged [10, 18, 48, 49, 55, 57, 78, 95, 99, 124, 128, 129]. Third, the adversary could directly read information from the respective buses to obtain details about the DNN [55, 56, 139]. In addition, if in-memory computing architecture is used to implement DNNs then these are also susceptible to reverse engineering [106]. Last, timing analysis can be performed in software by observing the time it takes to execute a query to the target DNN [33, 34]. Therefore, if the adversary has more knowledge and access to the target system, other SCA approaches could be leveraged to extract the DNN as well. For instance, once the architecture is extracted using the methods mentioned in Section 5, software based memory access pattern attacks [39] can be used to extract the parameters. However, combined software and hardware approaches would assume a much stronger attacker, that can access the same device physically and locally. On the other hand, the results of the combined approaches are potentially superior.

Algorithm-based model stealing. Model extraction is not exclusively possible with SCA approaches, one could also leverage algorithm-based stealing to extract the network [19, 20, 94, 119]. We provided connections between algorithm-based model stealing and physical SCA attacks in Section 8.

Related surveys. No systematization paper provides a comprehensive overview of research on model extraction and input recovery techniques through physical SCA. Existing works discuss the general approach towards model extraction and related defenses against it [61, 79, 84, 94]. However, due to the more generic approach, these works fail to mention the specific attack scenario and limitations other than those noted. On the other hand, use-case-specific surveys exist that discuss model extraction, e.g., in reinforced learning [59], edge-deployed [60], and cloud-deployed neural networks [40]. This work extends the research field by discussing model extraction and input recovery attacks, specifically for the case in which physical SCA approaches are leveraged.

10 Conclusion

Our paper systematizes findings on physical SCA approaches towards model extraction and input recovery, and provides a framework that breaks down current SCA attacks into different stages. We provide a generic and specific threat model, and a taxonomy of relevant previous works on this topic. We offer a detailed analysis and perform experiments to illustrate potential shortcomings in common SCA practices towards model extraction and input recovery. Future works may benefit from the presented threat models, which increase the generalization and robustness of SCA approaches. Further directions could aim at making the SCA approaches more efficient, targeting DNNs with a larger architecture and more parameters.

Acknowledgments

This work was funded by the Dutch Research Council (NWO) through the PROACT project (NWA.1215.18.014) and TTW PREDATOR project 19782. Dirk Lauret was supported by Adnovum AG throughout this work.

References

[1] Adversarial threat landscape for artificial-intelligence systems. https://github.com/mitre/advmlthreatmatrix. Accessed: 2024-06-13.

[2] Cyberattacks against machine learning systems are more common than you think. https://www.microsoft.com/en-us/security/blog/2020/10/22/cyberattacks-against-machine-learning-systems-are-more-common-than-you-think/. Accessed: 2024-06-13.

[3] Deep learning. https://developer.nvidia.com/deep-learning. Accessed: 2024-06-13.

[4] Jetson modules. https://developer.nvidia.com/embedded/jetson-modules. Accessed: 2024-06-13.

[5] Models and pre-trained weights. https://pytorch.org/vision/stable/models.html. Accessed: 2024-06-13.

[6] Navigate threats to AI systems through real-world insights. https://atlas.mitre.org/. Accessed: 2024-06-13.

[7] Privacy preserving machine learning innovation. https://www.microsoft.com/en-us/research/group/privacy-preserving-machine-learning-innovation/. Accessed: 2024-06-13.

[8] Xilinx DPU for convolutional neural network. https://www.xilinx.com/products/intellectual-property/dpu.html. Accessed: 2024-06-13.

[9] Xilinx ZCU 104 user guide. https://docs.xilinx.com/v/u/en-US/ug1267-zcu104-eval-bd. Accessed: 2024-06-13.

[10] M. Alam and D. Mukhopadhyay. How secure are deep learning algorithms from side-channel based reverse engineering? In *Annual Design Automation Conference*, 2019.

[11] L. Batina, S. Bhasin, D. Jap, and S. Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *USENIX Security Symposium*, 2019.

[12] L. Batina, S. Bhasin, D. Jap, and S. Picek. Poster: Recovering the input of neural networks via single shot side-channel attacks. In *ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[13] D. Boneh, R. DeMillo, and R. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology*, 1997.

[14] J. Breier, D. Jap, X. Hou, and S. Bhasin. A desynchronization-based countermeasure against side-channel analysis of neural networks. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, 2023.

[15] P. Brey. Ethical aspects of facial recognition systems in public places. *Journal of Information, Communication and Ethics in Society*, 2(2):97–109, 2004.

[16] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems*, 2004.

[17] T. Buzer. Deep neural network reverse engineering through side channel information leakage, 2022. https://catalogue-bibli.ec-lyon.fr/cgi-bin/koha/opac-detail.pl?biblionumber=134213.

[18] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2018.

[19] I. Canales-Martínez, J. Chávez-Saab, A. Hambitzer, F. Rodríguez-Henríquez, N. Satpute, and A. Shamir. Polynomial time cryptanalytic extraction of neural network models. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2024.

[20] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[21] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *Workshop on Frontiers in Handwriting Recognition*, 2006.

[22] Ł. Chmielewski and L. Weissbart. On reverse engineering neural network implementation on GPU. In *Applied Cryptography and Network Security Workshops*, 2021.

[23] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016.

[24] J.-S. Coron and L. Goubin. On boolean and arithmetic masking against differential power analysis. In *Cryptographic Hardware and Embedded Systems*, 2000.

[25] D. Das, M. Nath, S. Ghosh, and S. Sen. Killing EM side-channel leakage at its source. In *IEEE International Midwest Symposium on Circuits and Systems*, 2020.

[26] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli. Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks. In *USENIX Security Symposium*, 2019.

[27] G. Dong, P. Wang, P. Chen, R. Gu, and H. Hu. Floating-point multiplication timing attack on deep neural network. In *IEEE International Conference on Smart Internet of Things*, 2019.

[28] A. Dubey, R. Cammarota, and A. Aysu. Bomanet: boolean masking of an entire neural network. In *International Conference on Computer-Aided Design*, 2020.

[29] A. Dubey, R. Cammarota, and A. Aysu. MaskedNet: The first hardware inference engine aiming power side-channel protection. In *IEEE International Symposium on Hardware Oriented Security and Trust*, 2020.

[30] A. Dubey, R. Cammarota, V. Suresh, and A. Aysu. Guarding machine learning hardware against physical side-channel attacks. *Journal on Emerging Technologies in Computing Systems*, 18(3):1–31, 2022.

[31] A. Dubey, E. Karabulut, A. Awad, and A. Aysu. High-fidelity model extraction attacks via remote power monitors. In *IEEE International Conference on Artificial Intelligence Circuits and Systems*, 2022.

[32] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 2022.

[33] V. Duddu and D. V. Rao. Quantifying (hyper) parameter leakage in machine learning. In *IEEE International Conference on Multimedia Big Data*, 2020.

[34] V. Duddu, D. Samanta, D. V. Rao, and V. Balas. Stealing neural networks via timing side channels, 2018.

[35] C. Dwork. Differential privacy. In *International Colloquium on Automata, Languages, and Programming*, 2006.

[36] F. Elibol, U. Sarac, and I. Erer. Realistic eavesdropping attacks on computer displays with low-cost and mobile receiver system. In *IEEE European Signal Processing Conference*, 2012.

[37] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *USENIX Security Symposium*, 2014.

[38] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis:concrete results. In *Cryptographic Hardware and Embedded Systems*, 2001.

[39] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal. DeepTheft: Stealing DNN model architectures through power side channel. In *IEEE Symposium on Security and Privacy*, 2024.

[40] X. Gong, Q. Wang, Y. Chen, W. Yang, and X. Jiang. Model extraction attacks and defenses on cloud-based machine learning models. *IEEE Communications Magazine*, 58(12):83–89, 2020.

[41] C. Gongye, Y. Luo, X. Xu, and Y. Fei. Side-channel-assisted reverse-engineering of encrypted DNN hardware accelerator IP and attack surface exploration. In *IEEE Symposium on Security and Privacy*, 2024.

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

[43] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[46] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018(3):123–142, 2018.

[47] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

[48] S. Hong, M. Davinroy, Y. Kaya, D. Dachman-Soled, and T. Dumitraş. How to 0wn nas in your spare time. In *International Conference on Learning Representations*, 2020.

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

[50] Z. Hongxin, H. Yuewang, W. Jianxin, L. Yinghua, and Z. Jinling. Recognition of electro-magnetic leakage information from computer radiation with SVM. *Computers & Security*, 28(1-2):72–76, 2009.

[51] P. Horvath, L. Chmielewski, L. Weissbart, L. Batina, and Y. Yarom. BarraCUDA: GPUs do leak DNN weights, 2023.

[52] P. Horvath, L. Chmielewski, L. Weissbart, L. Batina, and Y. Yarom. CNN architecture extraction on edge GPU. In *Applied Cryptography and Network Security Workshops*, 2024.

[53] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[54] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales. When face recognition meets with deep learning: An evaluation of convolutional neural networks for face recognition. In *IEEE International Conference on Computer Vision Workshops*, 2015.

[55] X. Hu, L. Liang, L. Deng, S. Li, X. Xie, Y. Ji, Y. Ding, C. Liu, T. Sherwood, and Y. Xie. Neural network model extraction attacks in edge devices by hearing architectural hints. 2019.

[56] 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 *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[57] W. Hua, Z. Zhang, and G. E. Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Annual Design Automation Conference*, 2018.

[58] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[59] I. Ilahi, M. Usama, J. Qadir, M. U. Janjua, A. Al-Fuqaha, D. Hoang, and D. Niyato. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE Transactions on Artificial Intelligence*, 3(2):90–109, 2021.

[60] M. Isakov, V. Gadepally, K. M. Gettings, and M. Kinsy. Survey of attacks and defenses on edge-deployed neural networks. In *IEEE High Performance Extreme Computing Conference*, 2019.

[61] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In *USENIX Security Symposium*, 2020.

[62] B. Jayaraman and D. Evans. Evaluating differentially private machine learning in practice. In *USENIX Security Symposium*, 2019.

[63] R. Joud, P. Moëllic, S. Pontié, and J. Rigaud. A practical introduction to side-channel extraction of deep neural network parameters. In *International Conference on Smart Card Research and Advanced Applications*, 2022.

[64] R. Joud, P. Moellic, S. Pontie, and J. Rigaud. Like an open book? Read neural network architecture with simple power analysis on 32-bit microcontrollers. In *International Conference on Smart Card Research and Advanced Applications*, 2023.

[65] N. 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 *International Symposium on Computer Architecture*, 2017.

[66] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta. Model extraction warning in MLaaS paradigm. In *Annual Computer Security Applications Conference*, 2018.

[67] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten. CrypTen: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 2021.

[68] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology*, 1996.

[69] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology*, 1999.

[70] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009. https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf.

[71] M. Kuhn and R. Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In *International Workshop on Information Hiding*, 1998.

[72] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):255–258, 1995.

[73] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[74] E. Li, Z. Zhou, and X. Chen. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Workshop on Mobile Edge Communications*, 2018.

[75] G. Li, M. Tiwari, and M. Orshansky. Power-based attacks on spatial DNN accelerators. *ACM Journal on Emerging Technologies in Computing Systems*, 18(3):1–18, 2022.

[76] S. Liang, Z. Zhan, F. Yao, L. Cheng, and Z. Zhang. Clairvoyance: Exploiting far-field EM emanations of GPU to "see" your DNN models through obstacles at distance. In *IEEE Security and Privacy Workshops*, 2022.

[77] G. Litjens, T. Kooi, B. Bejnordi, A. Setio, F. Ciompi, M. Ghafoorian, J. Van Der Laak, B. Van Ginneken, and C. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, 2017.

[78] Y. Liu and A. Srivastava. Ganred: GAN-based reverse engineering of DNNs via cache side-channel. In *ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2020.

[79] Y. Liu, M. Zuzak, D. Xing, I. McDaniel, P. Mittu, O. Ozbay, A. Akib, and A. Srivastava. A survey on side-channel-based reverse engineering attacks on deep neural networks. In *International Conference on Artificial Intelligence Circuits and Systems*, 2022.

[80] Z. Liu, N. Samwel, L. Weissbart, Z. Zhao, D. Lauret, L. Batina, and M. Larson. Screen gleaning: A screen reading tempest attack on mobile devices exploiting an electromagnetic side channel. *Network and Distributed System Security Symposium*, 2021.

[81] H. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng. Can one hear the shape of a neural network?: Snooping the GPU via magnetic side channel. In *USENIX Security Symposium*, 2022.

[82] S. Maji, U. Banerjee, and A. Chandrakasan. Leaky Nets: Recovering embedded neural network models and inputs through simple power and timing side-channels—attacks and defenses. *IEEE Internet of Things Journal*, 8(15):12079–12092, 2021.

[83] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[84] M. Méndez Real and R. Salvador. Physical side-channel attacks on embedded neural networks: A survey. *Applied Sciences*, 11(15):6790, 2021.

[85] V. Meyers, D. Gnad, and M. Tahoori. Reverse engineering neural network folding with remote FPGA power analysis. In *International Symposium on Field-Programmable Custom Computing Machines*, 2022.

[86] V. Meyers and M. Tahoori. Power side-channel attacks and defenses for neural network accelerators. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2023.

[87] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, et al. FP8 formats for deep learning, 2022.

[88] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, 2017.

[89] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier. Remote power side-channel attacks on BNN accelerators in FPGAs. In *Design, Automation & Test in Europe Conference & Exhibition*, 2021.

[90] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh. Rendered insecure: GPU side channel attacks are practical. In *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[91] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

[92] M. Naseri and N. Beaulieu. Fast simulation of additive generalized gaussian noise environments. *IEEE Communications Letters*, 24(8):1651–1654, 2020.

[93] Y. Nozaki and M. Yoshikawa. Shuffling countermeasure against power side-channel attack for MLP with software implementation. In *International Conference on Electronics and Communication Engineering*, 2021.

[94] D. Oliynyk, R. Mayer, and A. Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Computing Surveys*, 55(14):1–41, 2023.

[95] B. Olney and R. Karam. Bits to BNNs: Reconstructing FPGA ML-IP with joint bitstream and side-channel analysis. In *IEEE International Symposium on Hardware Oriented Security and Trust*, 2023.

[96] OpenAI. GPT-4 Technical Report, 2023.

[97] T. Orekondy, B. Schiele, and M. Fritz. Knockoff Nets: Stealing functionality of black-box models. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[98] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, 2017.

[99] K. Patwari, S. Hafiz, H. Wang, H. Homayoun, Z. Shafiq, and C. Chuah. DNN model architecture fingerprinting attack on cpu-gpu edge devices. In *IEEE European Symposium on Security and Privacy*, 2022.

[100] L. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[101] E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. In *Advances in Cryptology*, 2013.

[102] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, 2016.

[103] M. Razzak, S. Naz, and A. Zaib. Deep learning for medical image processing: Overview, challenges and the future. *Classification in BioApps: Automation of Decision Making*, pages 323–350, 2018.

[104] F. Regazzoni, S. Bhasin, A. Pour, I. Alshaer, F. Aydin, A. Aysu, V. Beroulle, G. Di Natale, P. Franzon, D. Hely, et al. Machine learning and hardware security: Challenges and opportunities. In *International Conference on Computer-Aided Design*, 2020.

[105] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[106] S. Sayyah Ensan, K. Nagarajan, M. Khan, and S. Ghosh. Scare: Side channel attack on in-memory computing for reverse engineering. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(12):2040–2051, 2021.

[107] S. Sharma, U. Kamal, J. Tong, T. Krishna, and S. Mukhopadhyay. SNATCH: Stealing neural network architecture from ML accelerator in intelligent sensors. In *IEEE SENSORS*, 2023.

[108] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, 2017.

[109] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[110] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur. X-vectors: Robust DNN embeddings for speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018.

[111] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

[112] V. Sze, Y. Chen, T. Yang, and J. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[113] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[114] G. Takatoi, T. Sugawara, K. Sakiyama, and Y. Li. Simple electromagnetic analysis against activation functions of deep neural networks. In *Applied Cryptography and Network Security Workshops*, 2020.

[115] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019.

[116] M. Thu, M. Méndez Real, M. Pelcat, and P. Besnier. You only get one-shot: Eavesdropping input images to neural network by spying soc-FPGA internal bus. In *International Conference on Availability, Reliability and Security*, 2023.

[117] 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 *International Symposium on Field-Programmable Custom Computing Machines*, 2021.

[118] K. Tiri and I. Verbauwhede. Charge recycling sense amplifier based logic: Securing low power security ICs against DPA [differential power analysis]. In *European Solid-State Circuits Conference*, 2004.

[119] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security Symposium*, 2016.

[120] C. Van Der Malsburg. Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. In *Brain Theory*, 1986.

[121] W. Van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[122] R. Van Sloun, R. Cohen, and Y. Eldar. Deep learning in ultrasound imaging. *Proceedings of the IEEE*, 108(1):11–29, 2019.

[123] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[124] Z. Wang, X. Zeng, X. Tang, D. Zhang, X. Hu, and Y. Hu. Demystifying arch-hints for model extraction: An attack in unified memory system, 2022.

[125] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Annual Computer Security Applications Conference*, 2018.

[126] S. Wolf, H. Hu, R. Cooley, and M. Borowczak. Stealing machine learning parameters via side channel power attacks. In *IEEE Computer Society Annual Symposium on VLSI*, 2021.

[127] Y. Xiang, Z. Chen, Z. Chen, Z. Fang, H. Hao, J. Chen, Y. Liu, Z. Wu, Q. Xuan, and X. Yang. Open DNN box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2717–2721, 2020.

[128] M. Yan, C. W. Fletcher, and J. Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *USENIX Security Symposium*, 2020.

[129] D. Yang, P. J. Nair, and M. Lis. Huffduff: Stealing pruned DNNs from sparse accelerators. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[130] Y. Yang, C. Lin, Q. Li, Z. Zhao, H. Fan, D. Zhou, N. Wang, T. Liu, and C. Shen. Quantization aware attack: Enhancing transferable adversarial attacks by model quantization. *IEEE Transactions on Information Forensics and Security*, 2024.

[131] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *IEEE Computer Security Foundations Symposium*, 2018.

[132] V. Yli-Mäyry, A. Ito, N. Homma, S. Bhasin, and D. Jap. Extraction of binarized neural network architecture and secret parameters using side-channel information. In *IEEE International Symposium on Circuits and Systems*, 2021.

[133] K. Yoshida, T. Kubota, S. Okura, M. Shiozaki, and T. Fujino. Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator. In *IEEE International Symposium on Circuits and Systems*, 2020.

[134] 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*, 104(1):152–161, 2021.

[135] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin. DeepEM: Deep neural networks model recovery through EM side-channel information leakage. In *IEEE International Symposium on Hardware Oriented Security and Trust*, 2020.

[136] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015.

[137] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. Al Faruque. Stealing neural network structure through remote FPGA side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.

[138] Z. Zhao, P. Zheng, S. Xu, and X. Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019.

[139] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu. Hermes attack: Steal DNN models with lossless inference accuracy. In *USENIX Security Symposium*, 2021.