SoK: Model Reverse Engineering Threats for Neural Network Hardware

Seetal Potluri
Department of Electrical and Computer Engineering
University at Albany, SUNY, NY 12208
Email: spotluri@albany.edu

Farinaz Koushanfar
Department of Electrical and Computer Engineering
University of California San Diego, CA 92093
Email: farinaz@ucsd.edu

***Abstract*—There has been significant progress over the past seven years in model reverse engineering (RE) for neural network (NN) hardware. Although there has been systematization of knowledge (SoK) in an overall sense [1], [2], however, the treatment from the hardware perspective has been far from adequate. To bridge this gap, this paper systematically categorizes the types of NN hardware used prevalently by the industry/academia, and also the model RE attacks/defenses published in each category. Further, we sub-categorize existing NN model RE attacks based on different criteria including the degree of hardware parallelism, threat vectors like side-channels, fault-injection, scan-chain attacks, system-level attacks, type of asset under attack, the type of NN, *exact* versus *approximate* recovery, etc.**

We make important technical observations and identify key open research directions. Subsequently, we discuss the state-of-the-art defenses against NN model RE, identify certain categorization criteria, and compare the existing works based on these criteria. We note significant qualitative gaps for defenses, and suggest recommendations for important open research directions for protection of NN models. Finally, we discuss limitations of existing work in terms of the types of models where security evaluation or defenses were proposed, and suggest open problems in terms of protecting practically expensive model IPs.

1. Introduction

Artificial intelligence (AI) is currently widely used in applications related to many aspects of life including weather prediction, transportation, social networking, health, advertising, and more recently in chatbots. Within this broad field, machine learning (ML) is a sub-field that uses algorithms to analyze large amounts of data, learn from the insights, and then make informed decisions. Traditional ML required considerable domain expertise and significant manual effort to perform the feature extraction step that transformed the raw data into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Deep-learning (DL) is a sub-field of ML that allows a machine to be fed with raw data, which will then be able to perform automatic feature extraction [3]. DL was found to be very successful in extracting useful knowledge from high-dimensional data in many practical problems in image/speech/text/video/audio recognition, that resisted the AI community for many decades. This relationship between AI, ML, and DL is highlighted in Figure 1. DL constitutes deep neural networks (DNNs) with input, hidden, and output layers. Different types of DL architectures exist: fully connected feed-forward networks (FCNs) or multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), autoencoders, generative adversarial networks (GANs), and transformers being some of the most popular/prominent ones as shown in Figure 1.

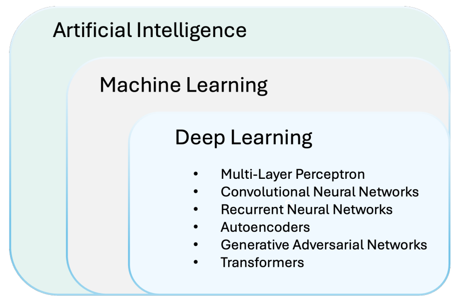

Figure 1. Classification of AI algorithms.

Coming to hardware implementations of these algorithms, they are twofold: (a) low-cost (area/power/performance) devices that leverage serial implementations and (b) hardware accelerators that leverage powerful systolic arrays for highly parallel execution to achieve high throughput. The first class of NN hardware is used in low-power IoT/edge applications that demand a low hardware footprint. DL algorithms are indispensable with these edge devices used in sensors, actuators, etc. [4]. Moreover, these devices are typically battery-constrained and hence use low-power processors that perform serial execution.

The second class of devices are used in compute-intensive applications and sometimes have hard real-time performance requirements, which cannot be met by serial NN hardware. To meet these performance goals, there has been significant progress made in NN hardware accelerator development within both the industry and academia. These accelerators leverage parallel execution to maintain high throughput. This is also reflected in the proliferation of AI hardware startups, corresponding funding, and AI companies increasingly diversifying into chip design e.g. OpenAI.

Recently, *SambaNova Systems* announced a new milestone in accelerating AI workloads, achieving as high as 1000 tokens per second with the `Llama-3` 8 billion parameter large language model (LLM) [5], which is also a variant of NN. Considering the costs of design and development of the NN models that run on these devices, protecting their *intellectual property (IP)* becomes a critical concern. For example, it is estimated that training the `GPT-4` model costs over $100 million [6]. As a result, there have been numerous model IP RE attacks published over the last few years. Apart from IP theft, since the model is a strong function of the customer's private data, data privacy is an orthogonal concern [5]. The number of published papers in hardware-based model RE attacks and defenses are shown in Figure 2. It can be seen that there is a systematic rise in both attacks and defenses over the years, although not monotonic.

1.1. Software Attacks

In software attacks, one prevalent area of research is teacher-student models. This approach entails creating a dataset based on the teacher model and using it to train student models [2], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. The student model may acquire a biased understanding, lacking the comprehensive intelligence of the teacher model, as the attacker typically lacks access to the actual dataset used to train the teacher model. Additionally, defenses against adversarial attacks can further mislead the student, hindering their ability to accurately learn from the teacher model. Another type of software attack involves cryptanalytic RE of neural networks (NNs). Attackers manipulate inputs to identify input-output relationships, allowing them to identify critical points within the activation regions [18], [19], [20], [21], [22] of ReLU networks. However, these attacks are usually limited to networks with two or three layers due to their complexity.

1.2. Hardware Attacks

Since only the query-label pairs are available while intermediate states i.e. after each layer, are unavailable through software, extracting each model parameter becomes a mathematically daunting task. On the other hand, such internal states are accessible through hardware side-channels, scan-chains, etc., making hardware attacks many orders of magnitude more powerful both in terms of queries and RE accuracy [23], [24]. These attacks have been demonstrated on various forms of hardware, which can be broadly categorized into four types: (a) central processing units (CPUs), which include both single-core and multi-core ones, (b) graphics processing units (GPU), (c) field-programmable gate arrays (FPGAs), and (d) application specific integrated circuit (ASIC) accelerators. In terms of computation, some of them are limited to *serial* execution, while others are capable of *parallel* execution. Figure 3 shows the distribution of existing model RE works (both attacks and defenses included) in terms of serial or parallel execution. According to this plot, the majority of the existing work focus on serial execution. This observation along with the fact that today's real-world NN hardware products are dominated by accelerators that perform parallel execution, shows that **there is a significant gap between what is done and what needs to be done.**

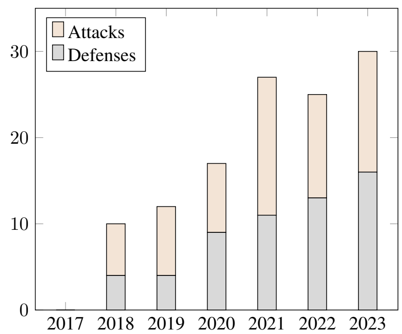

Figure 2. Literature over the years on NN hardware attacks/defenses.

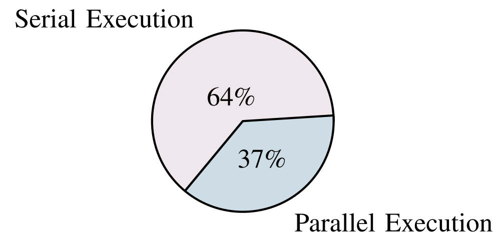

Figure 3. Distribution of works with respect to serial/parallel execution.

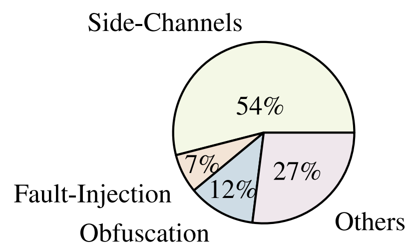

Figure 4. Distribution of different classes of attacks/defenses.

Orthogonally, the threat vectors themselves can be categorized as (a) side channels, (b) fault injection, (c) scan-chain attacks, and (d) other system-level attacks. Figure 4 shows the distribution of the attack vectors (both attacks and defenses) into these different categories. It is interesting to note that **side channels dominate** the ensemble of existing works, and hence we shall provide special treatment to side channels later in the paper.

1.3. Contributions

There exist rigorous surveys and systematization of knowledge (SoK) works for software-based NN model RE [2], [25], but there is little to almost no systematization of knowledge (SoK) for hardware-based NN model RE. Existing surveys [2], [26], [27], [28], [29] on NN model RE through hardware do not systematically analyze the vulnerabilities and defenses. Each of them is either not comprehensive, focuses exclusively on hardware side-channels, or is restricted to certain classes of attacks/defenses or only to serial implementations. Our main contributions are as follows:

- We categorize existing hardware-based NN model RE attacks. We also provide detailed taxonomy based on the underlying hardware type.
- We identify side channels as the dominant research direction pursued so far. We identify the gaps in the existing work on security evaluation and suggest open research directions.
- Since the future is converging towards ASIC accelerators meant for parallel execution, we identify the gap between this trend and the existing work that is dominated by serial implementations. We provide recommendations on how to bridge this gap.
- We also categorize the huge volume of existing hardware-based defenses, and provide a detailed comparison between different works in terms of different figures of merit.
- Although there is a huge volume of defense papers, we show there are significant qualitative gaps. Based on this, we suggest important open research directions for protection of NN hardware.

2. Neural Networks

A neural network (NN) can be described as a series of functional transformations, where the architecture and parameters completely describe the NN model. A training set is used to train/tune the parameters of an adaptive NN model. These parameters are typically called *weights* and *biases*. The training is typically done using the backpropagation algorithm [3]. Once the training is complete, the network is used to perform classification or regression. This work focuses on NN models for classification. NNs are organized into layers, the first called *input layer*, the last one the *output layer*, and the in-between ones as the *hidden layers*. There are different types of neural networks, some of the popular ones include:

2.1. Multi-Layer Perceptrons (MLPs)

It is a fully-connected feed-forward neural network with multiple layers of neurons. Neurons of one layer are fully connected to the neurons of the next as well as the previous layers.

2.2. Convolutional Neural Networks (CNNs)

They are a class of NNs, that have shown superior performance over classical ML approaches for image recognition [30]. It is a special kind of NN built for processing information with a grid-like structure, and typically consisting of (a) *convolutional layers* for preserving the relationship between the inputs while extracting the underlying features; (b) *pooling layers* to produce feature maps that are invariant to small changes in the input data [31]; (c) *fully connected* or *dense layers* at the end to predict the probability distributions of the input over different classes. Others sometimes include *batch normalization* and *dropout layers*.

2.3. Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are suitable for working with sequential data to solve problems where there is a temporal dependency. Unlike convolutional neural networks (CNNs) and multilayer perceptrons (MLPs), a RNN possesses a memory attribute. The memory allows the RNN to use previously seen values with the current input to predict the following event. LSTM is a popular example that uses RNN-style architecture. In RNNs, the weights are adjusted by training with backpropagation through time (BPTT), a variation on backpropagation seen in MLPs.

2.4. Autoencoders

Stacked auto-encoders are NNs with many layers trained by following a very specific procedure. This procedure consists of training each layer independently, using the output of the previous layer as input for the current one. Each layer is composed of an encoder and a decoder, both being a dense layer (i.e. fully connected layer).

2.5. Generative Adversarial Networks

They are a special class of NNs that can be used to generate data, consisting of two networks, *generator* and *discriminator*. The *generator* learns to create samples that match the training sample distribution, while the *discriminator* learns to discriminate between them [2].

2.6. Transformers

They are advanced NNs for working with sequential data based out of multi-head attention mechanism [32]. They have been shown to be of superior quality and require less training time than RNNs.

3. Neural Network Hardware

Neural network hardware architectures exist in various forms based on memory capacities, and power, performance, area (PPA) requirements, and depending on the end user application. For IoT nodes, which are area and battery constrained, serial implementations are most suitable. On the other hand, for performance constrained applications e.g. a fast moving autonomous car detecting a stop sign, area and energy-efficiency are less critical, so designers tend to provide great levels of parallellism.

3.1. Serial Implementations

These are typically single-core central processing units (CPUs). These devices could be used both at the cloud and at the edge. Examples at the edge include `Fitbit` that use ARM Cortex-M4 [33]. These are typically low-power microcontrollers which form a fair share of the current market with huge dominance in mobile applications, but also seeing rapid adoption in markets like IoT, automotive, virtual and augmented reality, etc. [33]. On the other hand, in cloud and datacenter applications, typically high-performance CPUs with large computing power are used.

3.2. Parallel Implementations

These hardware platforms enable parallel execution, and come in various forms: (a) multi-core systems that enable task-level parallelism; (b) graphics processing units (GPUs) that enable data-level parallelism; (c) field-programmable gate arrays (FPGAs) what provide user with the flexibility to modify hardware at runtime; and (d) systolic arrays, which provide tiled array of processing elements (PEs) for providing high-throughput for multiply-accumulate like operations.

3.2.1. Multi-Core. These are straightforward extensions of the CPUs discussed above, with multiple cores integrated on the same die. They enable multi-threaded execution and sometimes could integrate NN hardware accelerators on the same die [34].

3.2.2. Graphics Processing Units (GPUs). This is the dominant category of hardware to train and one of the dominant ones to run NN models, due to their massive parallelism and energy-efficiency. They are prevalently used for both neural architecture search (NAS) and training the model parameters using the backpropagation algorithm. Depending on their architecture, modern GPUs can be divided into integrated GPU-CPU architectures (monolithically integrated on the same die) and discrete GPUs which are connected to CPU via PCIe [35].

While integrated GPUs are more energy efficient, discrete GPUs are typically used for AI due to their ability for compute acceleration [35]. When such GPUs are on the cloud, there arises potential vulnerabilities due to adversaries sharing them with the victim. GPU kernels are typically accelerated using a software platform called compute unified device architecture (CUDA) which typically involves three tasks: (a) copies input data from main memory to GPU memory; (b) launches computational kernels on GPU; and (c) finally transfers the results from GPU memory back to the main memory. These GPUs are bandwidth bounded, hence the GDDR (graphics double data rate) memory is used to increase the memory bandwidth [36].

3.3. Field-Programmable Gate Arrays (FPGAs)

FPGA is a competing platform to a GPU, for the purpose of accelerating both neural network inference and training [37]. A modern FPGA SoC consists of a (a) programmable logic (PL) subsystem; and (b) a processor subsystem. The PL subsystem consists of an array of configurable logic blocks (CLBs), block random access memory (BRAM) units, digital signal processing (DSP) blocks, etc. As the name suggests, the PL subsystem can be reconfigured using a user-defined bitstream. On the other hand, the processor subsystem helps adding a layer of software, and the ability to control hardware through simple software level primitives, thereby adding great flexibility and ease of use. In the modern context, FPGAs also coming up with additional hardware to accelerate AI applications e.g. AI engine [38].

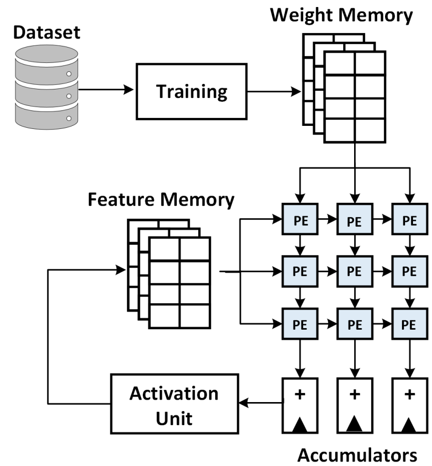

Figure 5. Hardware Accelerators for Neural Networks.

3.4. Systolic Arrays

There is currently a global race for the design of neural-network (NN) hardware accelerators with high-performance and low-power consumption, among most of the semiconductor companies [38], [39], [40], [41], [42], [43], [44], [45], [46], [47]. Figure 5 shows a tensor processing unit (TPU)-like systolic array based accelerator, which is representative of most of these accelerators albeit the differences in the low level details. The matrix multiply unit is at the heart of TPU, which helps perform layer wise computation within the NN. The weights and input features for this unit are read from independent partitions of an off-chip dynamic random access memory (DRAM) called weight and feature memories respectively. The complete accelerator architecture contains other components like host CPU, on-chip first-in first-out (FIFO) queues, on-chip buffers, host interfaces, and other control logic, which are skipped in Figure 5 for brevity.

4. Objective of the Adversary

The objective can be broadly categorised to be twofold: (a) breaching the confidentiality; and (b) breaching integrity. All the attacks published so far fall in one of these categories, or both in some cases.

4.1. Confidentiality

The adversary aims to RE (steal) the model IP through query access to the API combined with access to hardware internal states through various attacks discussed earlier. This typically involves targeting specific hardware components within the NN hardware, in order to extract the proprietary information including structure and/or the parameters of the model IP.

Definitions. The adversary aims to RE a *target model* denoted by f and uses input-response pairs (x, y). The attackers applies an input x to the oracle and obtains the prediction output y s.t. $f(x) = y$. If no other source of information is available, it is known as *black-box access*. On the contrary, if complete access to the architecture and model parameters is available, it is known as *white-box access*. The last scenario is somewhere in between, where the adversary has some additional information on what is going on inside the black-box beyond the outputs, it is known as *grey-box access*. We consider *black-box* and *grey-box* access scenarios in this paper. If the attacker obtains an approximate copy of the target model, it is denoted by $\hat{f}$.

Architecture/Structure: This refers to the model structure i.e. number of layers, layer types, number of neurons in each layer, etc. In case of MLPs, this translates to obtaining number of layers, number of neurons in each layer, type of activation function, etc. In case of CNNs, apart from this, types of each layer, kernel sizes per layer, etc. will also be needed. Basically, the type of information being retrieved depends on the underlying NN type.

Training Hyperparameters: Once the architecture is fixed/stolen, the adversary is interested to stealing the training hyperparameters, so the entity can train the substitute model in case of *approximate model extraction*. This corresponds to the set of hyperparameters used during training, like regularisation hyperparameters, batch size, optimization algorithm, etc. [2].

Model Parameters: Once architecture is fixed/stolen, the adversary is interested to extract the weights/bias values of all neurons in all the layers of the target model. In this case, the adversary is attempting at *identical behavior extraction* [2].

Objective. The adversary's objective can be of two types: (a) exact model extraction; or (b) approximate model extraction. In the first case, the attacker tries to reverse engineer the exact structure of the network in terms of layers, neuron count, hyperparameters, and exact values of the model parameters (weights/biases). In the latter case, the adversary is only interested in obtaining an approximate copy, which mimicks the behavior of the *target model*. In this case, there are again two sub-categories: (a) *high accuracy*, where the extracted model closely resembles the target model for the original data; and (b) *high fidelity*, where the extracted model is functionally equivalent to the target model i.e. predictions resembling with target model on any input [48].

In the field of logic deobfuscation, techniques like `AppSAT` [49] are used to solve the exploding query problem. Such techniques trade off query complexity with extraction accuracy. This gives the ability to handle hard instances of the problem by performing incremental analysis given a timeout. Although as discussed earlier, *approximate model recovery* methods exist, however techniques to trade-off accuracy with extraction effort are missing in the field of model RE. It is important to note that such security evaluation is critical, since such an adversary is more powerful than an exact adversary. It is because even for hard instances of the model RE problem, such an adversary has the ability to retrieve at least some partial information, that helps him to perform stealing.

> **Open Problem**: Exploring the tradeoff between extraction effort and RE accuracy for approximate model RE through NN hardware.

4.1.1. Adversary's capabilities. There are three main aspects when the adversary tries to perform model RE: knowledge about the target model, and available resources.

Knowledge: As discussed earlier, the attacker has black-box or gray-box access to the target model. In some works, they assumed that the attacker already has knowledge of the model architecture/structure, and proceed with the model parameter extraction.

Resources: Since we are interested in hardware attacks, the adversary has physical or remote access to the NN hardware target, either performing training or inference. In case of physical access, the adversary has access to side-channel probes, or other hardware measuring equipment to observe signals of interest, and measurement equipment like oscilloscope to capture signals of interest. Sometimes, signal conditioning/processing equipment like transceivers, impedance matchers, amplifiers, etc. will be used. In case of scan-chain attacks, access to the scan-chain infrastructure i.e. JTAG is needed. Such attacks will typically be thus exploited at third-party testing provider or in-field testing where scan-chain access is available along with system-level-test (SLT) capability.

4.1.2. Adversary's goals. In the case of *exact model extraction*, the extracted values must closely match the values of the target model. The most common metric to measure RE success is to minimize $max|\theta - \hat{\theta}|$, where $\theta \in f$.

Coming to *approximate model extraction*, as discussed earlier, there are *high-accuracy* and *high-fidelty* attacks. In the case of *high-accuracy* attacks, the adversary's goal is to minimize $max|f(x) - \hat{f}(x)|$, $\forall x \in T$, there T is the dataset used to train the target model. On the other hand, in the case of *high-fidelty* attacks, the adversary's goal is to make sure $f(x) = \hat{f}(x)$, $\forall x$.

4.1.3. Attack Success Rate. The adversary's success is measured in terms of how many NN architecture parameters or/and model parameters are successfully extracted, the extraction error, the number of queries used, the number of traces used in the case of side-channels, etc. In the case of remote attacks, the adversary has to pay per each query, so query minimization is an important measure of success. Even otherwise, similar to the field of physical side-channels, success rate is measured in terms of query/trace minimization to achieve the desired goal.

4.2. Integrity

This constitutes to the attacks that are popular in the adversarial machine learning regime. In software, there are two types of such attacks: (a) white-box setting; and (b) black-box setting. Since white-box settings are usually unrealistic in practice [50], [51], we focus only on black-box setting in this paper. In order to craft adversarial examples, the attacker can either makes assumptions about the victim model [52] or conduct *fingerprinting* [50] to infer more information required about the victim model.

Sometimes, the adversary exploits teacher-student models based on some openly available information in a transfer learning setting to obtain this information [50]. All the definitions, goals, objectives, capabilities, etc. defined earlier in section 4.1 hold good here as well, because integrity attacks also aim to retrieve the model structure and parameters as accurately as possible.

5. Categorization Criteria for Attacks

The existing work on model reverse engineering (RE) can be categorised based on different criterion, depending on the angle of view that the researcher is interested in. The categorization points include *threat vector*, *the underlying end-user application platform*, *the hardware type*, *neural network type*, *asset*, *phase of attack*, etc.

5.1. Threat Vector

The different threat vectors include *side channels*, *fault injection*, *system-level attacks*, and *scan-chain attacks*.

5.2. Platform

Depending on the application, the target platform could be at the cloud or at the edge. Some of the works look at machine learning as a service (MLaaS) services with neural networks at the backend. For such cases, the target platform is the cloud. On the other hand, in case of physical side-channels, the target device needs physical probing, which cannot be executed with a cloud platform. In such cases, the target platform will be at the edge.

5.3. Hardware type

The attack and defense strategies are highly influenced by the underlying hardware architecture. For example, physical side-channels are generally ineffective for hardware accelerators that execute highly concurrent computation due the background noise, thus other styles of attack are preferred. As discussed earlier, we categorise based on CPU, GPU, FPGA, and ASIC accelerators.

5.4. Neural Network type

The attack procedure depends on the underlying NN type, which highly influences the kind of architecture, hyperparameters, etc. that need to be reverse-engineered.

5.5. Type of asset being stolen

There are two primary categories: (a) architecture stealing; and (b) model parameter stealing. Existing works try to steal either one of these two assets or in some special cases, both. Figure 6 shows the distribution of existing works into these two categories. It can be seen that majority of the works look at reverse engineering the model architecture/structure.

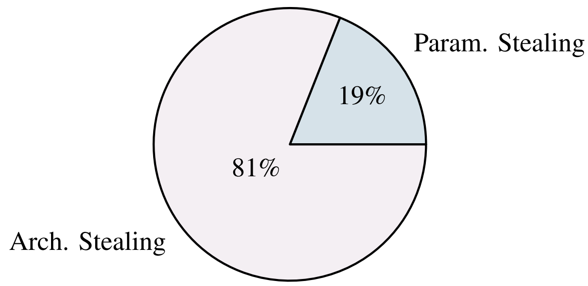

Figure 6. Distribution of attacks/defenses in terms of asset-under-attack.

5.6. Attack phase: training or inference?

Some selected works have launched the attack during the training phase, while most of the existing work have focused on the inference phase. It it difficult to steal during training because designers take several precautions, but if the adversary makes it possible, very valuable assets can be stolen during training. Coming to inference, anyone can launch most of the existing attacks, but it becomes very hard to steal during inference. For example, equivalent architecture extraction problem has been shown to be very hard due to the huge search space of the possible architectures [53].

6. Side Channel Attacks (SCAs)

We show our categorization of existing side-channel attacks (SCA) based on the criteria seen above in Table 1. A computing device interacts with its environment while executing different operations. Due to strong correlations of the data they operate on, to the physical properties of the device such as computation time, power consumption, electromagnetic (EM) radiation, etc., during such interactions, these devices leak sensitive information. Such physical leakages are popularly known as *side-channels* in the information security community. These *side-channels* can be qualitatively categorised into different categories, based on the way the confidential information is leaked. Some of the popular ones include microarchitectural, Trojan-based, physical, and remote side-channels.

6.1. Microarchitectural side-channels

As the name suggests, this category heavily relies on the microarchitecture of the implementation. As a result, it is critical to analyse microarchitectural side-channels individually for the four popular NN hardware choices: CPU, GPU, FPGA, and ASIC Accelerators. *Most of the time, these attacks could be exploited remotely because they do not need physical probing*.

6.1.1. CPU. It is well-known that micro-architecture side-channels that typically capture different memory or interconnect access patterns are very powerful in breaking crytosystems. Such side-channels have also been successfully applied to perform model RE for NNs. One popular sub-branch is *cache side-channels*, where it is assumed that the *spy/attacker* process is co-located with the *victim* process on

TABLE 1. SIDE CHANNEL ATTACKS

Target	Authors	Parallel Exec.	Microarch. SCA	Physical SCA	Remote SCA	Arch. Stealing	Param. Stealing	During Training	During Inf.	Distillation/TL/ Surrogates	NN type
CPU	Duddu et al. [53]	○	○	●	●	●	○	○	●	●	CNN
	Batina et al. [33]	○	○	●	○	●	●	○	●	○	MLP, CNN
	Yan et al. [54]	○	●	○	●	●	○	○	●	●	CNN
	Hong et al. [50]	○	●	○	●	●	○	●	●	●	CNN
	Jeong et al. [55]	○	●	○	●	●	●	○	●	○	MLP
	Liu et al. [56]	○	●	○	●	●	○	○	●	○	CNN
	Wolf et al. [57]	○	○	●	○	○	●	●	○	○	MLP
	Gao et al. [58]	○	○	●	●	●	○	○	●	●	CNN
	Gongye et al. [59]	○	○	●	●	○	●	○	●	○	MLP
	Maji et al. [60]	○	○	●	○	○	●	○	●	○	CNN
	Batina et al. [4]	○	○	●	○	●	●	○	●	○	MLP, CNN
	Gao et al. [34]	●	●	○	●	●	○	○	●	●	CNN
	Malan et al. [61]	○	○	○	●	●	○	○	●	○	CNN
	Ryu et al. [62]	○	○	○	●	●	○	○	●	●	CNN
	Won et al. [63]	○	●	○	●	●	●	○	●	●	CNN
GPU	Wei et al. [64]	●	●	○	●	●	○	●	○	○	CNN
	Hu et al. [65]	●	●	●	○	●	○	○	●	●	CNN
	Chmielewski et al. [66]	●	○	●	○	●	○	○	●	○	MLP, CNN
	Liang et al. [67]	●	○	●	●	●	○	○	●	○	CNN
	Wang et al. [68]	●	●	○	●	●	○	○	●	●	CNN
	Ahmadi et al. [69]	●	●	○	●	●	○	○	●	●	CNN
FPGA	Zhang et al. [70]	○	○	○	●	●	○	○	●	○	MLP, CNN
	Yu et al. [71]	○	○	●	○	●	●	○	●	●	BNN
	Yoshida et al. [72]	○	○	●	○	○	●	○	●	○	MLP
	Dubey et al. [73]	○	○	○	●	○	●	○	●	○	BNN
	Meyers et al. [74]	○	○	○	●	●	○	○	●	●	MLP, CNN
	Li et al. [75]	○	○	●	○	●	○	○	●	●	MLP
	Yli-Mäyry et al. [76]	○	○	●	○	●	●	○	●	○	BNN
	Dubey et al. [77]	○	○	●	○	○	●	○	●	○	BNN
ASIC Acc.	Hua et al. [78]	●	●	○	○	●	●	○	●	●	CNN
	Sharma et al. [79]	○	○	●	○	●	○	○	●	●	CNN
	Read et al. [80]	●	○	●	○	○	●	○	●	○	MLP
	Won et al. [81]	●	○	●	●	●	○	○	●	●	CNN
	Yang et al. [82]	●	●	○	○	●	○	○	●	●	CNN
	Gongye et al. [83]	○	○	●	○	●	●	○	●	○	CNN
	Yan et al. [84]	●	○	○	●	●	○	○	●	●	CNN
	Tian et al. [85]	●	○	○	●	●	○	○	●	○	CNN

● denotes "Yes", and ○ denotes "No", ✗denotes "Not Applicable.".

the same processor chip. One of the first *cache side-channel* works for model RE is *Cache Telepathy* [54] that exploits the extensive usage of geometrix matrix multiply (GEMM).

The attack proceeds in three steps: (a) using cache side-channel attack to reverse engineer the matrix dimensions; (b) perform detailed analysis of GEMM algorithms in ML frameworks to figure out the relationship between matrix parameters and the hyperparameters, and in this way reverse engineer a good initial solution to the model architecture; and (c) finally prune the candidate solutions to RE the architecture that is close to the target model. The authors evaluate both $Prime + Probe$ and $Flush + Reload$, two popular cache SCAs. Another cache side-channel attack on model RE is *DeepRecon* that uses $Flush + Reload$ to reverse-engineer the architecture and further build a meta-model that accurately fingerprints the architecture and family of the pretrained model in a transfer learning setting [50].

Another cache SCA is GANRED [56], an attack approach based on the generative adversarial nets (GAN) framework which utilizes cache timing side-channel information in the form of $Prime + Probe$ to accurately recover the structure of DNNs without memory sharing or code access. Unlike above cache SCAs, GANRED does not need any shared main memory segment between the victim and the attacker or analyze the DNN library codes on the server. GANRED uses an incremental approach to grow the retrieved structure using *generator*, *discriminator* and *validator* to measure side-channel information, victim comparisons, and pruning respectively.

Another popular sub-branch is *Meltdown* vulnerability [86] that allows memory read without access privileges by exploiting out-of-order execution. Researchers have expoited dictionary-type symbol tables in *Python* to identify the target memory address where the victim neural network is mapped, and subsequently exploit Meltdown to extract both the structure and model parameters of fully connected networks [55]. Orthogonally, researchers proposed `Tenet` [34], which demonstrate that malicious tenants in a multi-tenant multi-core system, where victim and spy are located on separate cores with a share memory channel. `Tenet` uses the fact the memory timing information in the shared memory channel exposes the knowledge of model architecture, which can be exploited by the malicious tenants to reverse-engineer the layer structure of neural networks.

6.1.2. GPU. Leaky DNN [64] looks the victim and the adversary the same GPU when training an NN, and exploit the context-switching side-channel to extract the structure of the NN, including layers and hyperparameters. This is one of the very few works across all platforms (CPU/GPU/FPGA/ASIC) that looks at RE vulnerability during training. `DeepSniffer` [65], on the other hand, learns the relation between extracted architectural hints like memory reads/write counts obtained by side-channel or bus snooping attacks and the details of the model architecture.

Unlike prior works, which look at incomplete model extraction in terms of layers and neurons details in the structure, this work looks at complete model extraction with run-time layer sequence identification, layer topology reconstruction, and dimension size estimation, and is robust to architecture-level noise. Finally, UMProbe [68] formally defines timing-sensitive architecture-level events, called the *Arch-hints*. Further, they identify existing *Arch-hints* are ineffective for unified memory (UM) management system for GPUs, and use their proposed formal definitions to develop a new attack surface for UM.

6.1.3. FPGA. Chandrasekar et al. [87] introduce a hardware Trojan to monitor memory side-channel information available on the Advanced eXtensible Interface (AXI) bus and leak it through the universal asynchronous receiver / transmitter (UART) port. This is the only work on microarchitectural side-channels on FPGA, while there has been significant work on physical side-channels and fault-injection on the FPGA. Since it is well-known that FPGAs are increasingly being used in the datacenters and virtualization technologies, similar to GPU, it is important to make sure if there are there are any memory or/and interconnect side-channels leaking sensitive information when executing NNs.

Open Problem: Exploring microarchitectural side-channels in cloud FPGAs and FPGAs in datacenters.

6.1.4. ASIC Accelerator. In Hua et al. [78], researchers have proposed an attack in the context of an NN hardware accelerator, where the accelerator is protected while the off-chip memory is not, similar to Intel SGX. In this context, they snoop the bus to observe the off-chip memory accesses like addresses and information on read/write operations, and use this information to successfully RE the structure and model parameter set of a CNN. Likewise, `HuffDuff` [82] was proposed that leverage the boundary effect present in the convolutional layers and the timing side channel of on-the-fly activation compression, to significantly prune the architecture search space during RE. Otherwise, there is not much work on microarchitectural side-channels for ASIC accelerators.

Since NN hardware is dominated by ASIC accelerators, and there is a global race amongst all the companies in the ASIC accelerator market, it is important to further explore microarchitectural side-channels in this context. Further, different companies consider different architectures. For example, Google's TPU looks at a more streamlined engine focused on multiply and accumulate operations. On the other hand, other companies like Tenstorrent rely on RISC-V based arrays, and AMD's AI engine is also based out of array of RISC CPUs. This leads to numerous threat vector possibilites, that need exploration.

Open Problem: Exploring microarchitectural side-channels in ASIC accelerators.

6.1.5. Other System-level attacks. Apart from microarchitectural side-channels, there are other system-level works explored in the literature. Researchers proposed DnD [88] that is able to recover a high-level representation of a DNN starting from its compiled binary code. It is the first compiler- and ISA-agnostic DNN decompiler, and does template matching to recover hyper-parameters, model parameters, and the overall DNN topology. Likewise, "Bits to BNNs" [89] is is the first method for NN model RE from the viewpoint of FPGA bitstream analysis and that further work is needed to improve security assurance for edge intelligence. For GPU, a new attack, called *Hermes* [35], was proposed, that snoops the *PCIe* bus between the host machine and GPU to RE both the structure and model parameters of an NN.

6.2. Trojan Side-Channels

A Trojan side-channel refers to a special kind of hardware Trojan that induces physical side-channels to leak secret information. Trojan side-channels have been extensively explored in the past for cryptosystems [90], [91], [92], [93]. Likewise, Trojan side-channel adversaries have also been proposed in the field of NN model RE in the context of versatile tensor accelerator (VTA) on Zynq UltraScale+ field programmable gate arrays (FPGA) [87]. This work aims at hyperparameter stealing in the context of CNNs. However, this is almost no other work in this direction.

Open Problem(s): Is it possible for adversaries to exploit hardware Trojan side-channels to steal model parameters? Can hardware Trojan side-channels be exploited for knowledge distillation like attacks?

6.3. Physical side-channels

Physical side-channels rely on the fact that all hardware implementations have unintended physical leakages. This includes power, electromagnetic (EM) emissions, timing, photonic emissions, scan-chains, etc. This class of attacks on NN hardware for model RE can be broadly categoriezed into two categories: (a) exact recovery; and (b) knowledge distillation/transfer-learning (TF)/surrogates.

Exact recovery: This again has two sub-categories: (a) the adversary has prior knowledge of the architecture/structure of the network, and interested to RE the model parameters; and (b) the adversary tries to perform RE on both the architecture/structure of the network and the model parameters.

Knowledge distillation: The adversary obtains exact or partial information on the architecture/structure from the side-channel, and trains this substitute model using a dataset which is again created using side-channel information.

Knowledge distillation is an easier model RE problem, compared to exact model RE problem. This is because in knowledge distillation, the adversary only looks at different hints on the architecture and training data, and is satisfied if the trained substitute model has decent accuracy. This is not the case with the exact model RE problem, because of the stringent constraints to obtain the exact structure in terms of layers, neurons per layer, and the exact model parameters.

Exact model RE works can be broadly categorized into two categories: (a) serial execution; and (b) parallel execution. Depending on the category, the approach to *physical side channel analysis* varies, although the challenges for *serial execution* is strictly a subset of the challenges for *parallel execution* due to the additional noise introduced in the latter case.

6.3.1. Serial Execution. Both simple power analysis (SPA), and differential power analysis (DPA) are well-known to successfully exploit the power side-channel to break both symmetric and asymmetric cryptosystems [94]. The key enabler in SPA is the identification of distinguishing power profiles for different instructions, while extraction the key bits becomes difficult due to small variations caused by the key bits amidst the high measurement noise. On the other hand, DPA extracts the statistical correlations between the individual secret key bits and the dynamic power dissipated by the microprocessor during the cryptographic computation over a large number of input cases, to successfully extract this secret key. Since power dissipation and EM radiation have a close mathematical relationship, researchers have extended DPA to EM side-channels, which is popularly known in the literature differential electro-magnetic analysis (DEMA) [95].

It is important to note that DPA/DEMA are successful on microprocessors that perform *serial execution*: in other words, the CPU executes only one instruction per clock cycle, thereby the power traces contain clear power patterns, revealing the instruction-level information in the temporal sense. In this case, the power consumed by each instruction is directly visible on the chip's power pin, thereby the correlation of the key bits is visible in the current consumed by the power pin, and not corrupted by background noise due to other instructions.

Researchers have extended the idea of extracting individual key bits to extracting individual weight bits in a binary neural network (BNN) and successfully exploited DPA to perform RE [73], [77], [96], [97], [98], [99]. Researchers have recently extended this to beyond BNNs to the more general arithmetic or multi-bit model parameter representations. Yoshida et al. [72] used correlation power analysis (CPA) to extract 8-bit fixed point model parameters of an MLP with partial success. Batina et al. [33] have extended DEMA to both 32-bit fixed and 32-bit floating point model parameters in both MLP and CNN, hence strongly establishing the universal applicability of DPA/DEMA on neural networks.

6.3.2. Parallel Execution. Unlike the serial execution scenario, in the case of parallel execution, multiple operations take place concurrently. In the TPU example discussed earlier, there is a 256×256 array of processing elements (PEs), all of them working simultaneously to accelerate the NN computations. In such cases, it is not possible to see observe direct correlations between an individual weight and the corresponding side-channel information in the temporal sense. Since all weights are operating simultaneously, *operation on one weight appears as noise to the other*, thereby there will be no correlations whatsover due to the extreme levels of parallelism. That is the reason most of physical side-channel works look at only architecture stealing or serial execution.

Open Problem: Exploring the application of physical side-channels to parallel execution scenarios. Is it possible to extract statistical correlations of the secret weights to the side-channel information?

Since most of the NN hardware today is not serial in nature, it is desirable to the adversary to convert the parallel execution scenario to the serial execution scenario so that all the rich source of literature on side-channels for NN model RE on serial execution hardware can be made directly applicable. To make this happen, the adversary intelligently chooses carefully crafted inputs, or quiescent operating points, also called the *Q-points*, which when applied the NN accelerator, executes operations only on one of the PEs, while all other PEs receives zeroed feature and weight inputs. This property is known as *linear constraint satisfaction* [23], [78], [83].

6.4. Remote Power Side-Channels

Except timing attacks, which are well-known to have the potent to be launched remotely [100], rest of the physical side-channels like power, EM, etc. were originally meant to be executed physically. However, after the first remote power attack on an FPGA [101], *remote power side-channels* have become very popular. These works attempt to repeat what could be done with *physical power side-channel* with *remote power side-channel* and report the success rate, resolution, signal-to-noise (SNR) ratio, etc. Due to common underlying cause, similar to *physical power side-channel*, *remote power side-channel* has the same issues related to parallel execution. Another interesting case study is *DeepTheft* [58], which looks at power side-channel on the cloud using running average power limit (RAPL) side-channel. Similar to physical side-channels, the challenge of extracting statistical correlations during parallel execution discussed above, persist for remote side-channels as well.

7. Fault Injection Attacks

Rakin et al. [102] proposed *HammerLeak* that performs rowhammer based fault injection [103] to infer weights and biases, when sharing the off-chip DRAM with the victim. The attack is stealthy and the adversary also trains a substitute based on the partial leakages due to bitflips, and thereby successfully performs model RE. They extend RAMBleed [104] which deploys rowhammer as a read side-channel, to DNNs. While RAMBleed requires the same page content in both aggressor rows, DNNs do not allow that. They address this issue by waving the requirement of having two duplicated copies of the victim page, and rather by substituting one victim page with attackers page while still being able to leak secret bits from the victim page.

Brier et al. [105] proposed a *sign bit flip fault (SNIFF) attack* that enables the reverse engineering by changing the sign of intermediate values. They specifically inject electromagnetic (EM) radition induced faults and target the deep-layer feature extractor networks produced by transfer learning, to recover the weights and biases of the last hidden layer.

Recently, Hector et al. [106] proposed a safe error attack (SEA), that relies on laser fault injection (LFI) using a bit-set fault model ($0 \rightarrow 1$, $1 \rightarrow 1$), that can perform a model extraction attack with an adversary having a limited access to training data to illegally train a substitute model. SEA enables to recover the MSB values of the victim model parameters, which in effect enables to efficiently constrain the substitute model training, with training data as low as 8%, while achieving high fidelity and near identical accuracy compared to the target/victim model. They focus on embedded DNNs on 32-bit microcontrollers, targeting IoT applications, making LFI feasible.

Other than this, the FI work for model RE is very limited. Most of the existing work on FI targets misclassification [107], [108], [109], [110], [111], [112], and not much for model RE attacks. There is a rich source of literature for fault injection attacks on cryptographic implementations. Unlike SCA, which has been successfully extended to NN model RE, FI is still an open direction.

Open Problem: Extending power, clock, EM, LFI, rowhammer, and other available fault injection attacks for cryptographic implementations to NN hardware for the purpose of model RE.

8. Scan-Chain Attacks

The scan-chain vulnerability exists at the third-party testing provider or during in-field testing. The adversary runs the classifier chip for a certain number of functional clock cycles, switches to test mode and dumps out the states of critical registers [23]. The switch between functional and debug modes also exists in the prior works on scan-chain attacks for cryptographic accelerators [113], but the timing of the clock and other control signals is unique to NN accelerators.

$$\boxed{\begin{array}{l} w_{n_1}^{\intercal}\mathbf{x} > b_{n_1} \\ w_j^{\intercal}\mathbf{x} \leq b_j, j \neq n_1 \end{array}}$$

Figure 7. Linear Constraint Satisfaction (LCS) system for a target neuron-n_1 in the first layer of a fully connected network, where $\mathbf{x}$ is the layer input, while $w_{n_1}^{\intercal}$ and b_{n_1} are the weight vector and bias values of neuron-n_1 respectively.

Constraint Satisfaction. Figure 7 shows the mathematical formulation of LCS system for neuron-n_1 in layer-1 of an MLP. It is important to note that for RE to be successful for layer-2, the model parameter set obtained after training should satisfy LCS for all neurons in layer-1 [23]. It was empirically demonstrated across a wide range of structures of different sizes and depths that this property is naturally satisfied for MLPs trained using the backpropagation algorithm [23]. While this is very useful from a practical perspective, it is not clear if this is always true. As a result, it is important to theoretically prove this is indeed always true, or the adversary has to come up with intelligent verification mechanisms for LCS satisfaction.

Open Problem: Why *linear constraint satisfaction* is naturally satisfied for fully connected networks? Orthogonally, is it possible to make sure/verify that the training output can be made to satisfy *linear constraint satisfaction*?

So far, LCS has been empirically demonstrated only for MLPs. Since CNNs are very popular and it is well-known that CNNs are used prevalently with NN accelerators, it is important that LCS be extended to CNNs as well. Likewise, it will interesting to check if this property for NNs that process sequential data like recurrent neural networks (RNNs).

Open Problem: Is *linear constraint satisfaction* also naturally satisfied for more complex networks like CNNs and RNNs?

In case of scan-chain attacks on cryptographic implementations [113], interrupting the hardware is relatively easy since the hardware is simple and has no layer of software. Thus, interrupting the hardware at a precise clock cycle to dump intermediate states is not very challenging. However, in case of NN hardware accelerators like Google's Tensor Processing Unit (TPU), the underlying hardware is complex and there is also a layer of software involved. This makes it extremely difficult to interrupt the NN hardware in real-time at a precise clock cycle, to dump out the internal states of the registers.

Open Problem: Is it possible to interrupt the NN hardware accelerator in real-time when the classification is is progress, at a precise clock cycle, to implement scan-chain attack?

In the case of physical side-channels, there are multiple companies like Riscure, ChipWhisperer, Sakura, Sasebo, etc. which provide products to validate the vulnerability. However, in case of scan-chain attacks at third-party testing provider or even in the field during in-field testing, although the vulnerability exists, there is no way for academic research to perform security evaluation in the lab environment and develop robust defenses.

Open Problem: Need to come up with platforms/products for scan-chain security evaluation, to promote research in this direction.

9. Categorization Criteria for Defenses

Similar to attacks, the defenses can be categorized based on different criterion, depending on the angle of view that the researcher is interested in.

9.1. Nature of Defense

The high-level categorization is done based on the nature of defense, which we identify as nine categories: (a) trusted execution environments (TEEs) and attestation, (b) hardware masking, (c) obfuscation, optimization, and perturbation (OOP) techniques, (d) application of cryptographic methods, (e) shuffling, dummy instruction/operation insertion, and randomization (SDR) defenses, (f) hardware-assisted defenses, (g) prediction poisoning methods, that thwart the adversary, (h) model fingerprinting methods , and (i) model compression defenses, depending on the nature of applied protection for the model IP.

9.2. Reactive or Proactive?

One of the categorization criteria is whether the defense is detection-based (reactive) or prevention-based (proactive). As we shall see later, most of the hardware defenses for model RE are prevention-based.

9.3. Impact on Actual Classifier Accuracy

Sometimes, companies have to pay a price to implement the defense in NN hardware/software co-system. While we are interested to protect the model, it is also important to understand how the changes made to the hardware/software due to the defense, will impact the classifier accuracy. As we shall see later, while some of the defenses have no impact on the classifier accuracy, some do incur a penalty which is the trade-off between security and the actual classifier accuracy.

9.4. Figures of Merit

Since we are interested in protecting NN models running on hardware, the defenses in most cases involve modifications to certain aspects of hardware/software. This can negatively impact the power, performance, area (PPA) of the underlying NN hardware. Thus, it is important to understand/analyze the tradeoffs between security and PPA figures of merit of the NN hardware.

10. Defenses

Table 2 shows the different defenses published in the literature, in terms of the categorization criteria above.

10.1. TEEs and Attestation

These techniques aim full or partial execution of the confidential computations inside a TEE like Intel SGX, ARM TrustZone, Sanctum, Graviton, etc., as well as device attestations for co-processors for providing access control. It has been shown that the full TEE-based DNN execution incurs $10\times$, partial TEE-based DNN execution incurs $\approx 2\times$ performance penalty [114], [156], while attestation techniques like DeepAttest [115], [156] incurs only 1.3% performance penalty. While these attestation techniques use watermarks to verify the legitimacy of the deployed DNN before allowing it to execute normal inference, they still cannot guarantee protection against physical side-channels or scan-chain attacks.

10.2. Hardware Masking

Masking defenses split inputs of all weight-dependent computations into two randomized shares, which are then independently processed and are reconstituted at the final step when the final output is generated. This is done so as to decorrelate the computations to the secret weights, to make physical side-channel attacks infeasible. This is further augmented with *hiding* techniques to decorrelate the sign bit from the input. There are numerous hardware masking defenses in the literature [77], [96], [97], [98], [99], [129], [144].

10.3. OOP Defenses

There are numerous model and input obfuscation defenses proposed for model RE [69], [123], [124], [126], [128], [131], [132], [133], [140], [144], [146], [151], [157], [158], that fall in this category. Likewise, works that use model optimization [134] to defend RE and model perturbation defenses also fall in this category. Although there are model perturbation defenses for software, there are currently no model perturbation defenses for hardware model RE.

Open Problem: Is it possible to perform secure training or perturb the models, so as to make the model more secure against model RE attacks?

10.4. Cryptographic Defenses

These are encryption based techniques that tries to protect model parameters from unprotected components, by encrypting them before they reach such components. Typically, these techniques incur a performance overhead.

TABLE 2. DEFENSES.

Defense Class	Authors	Detection	Prevention	Impact on Classifier Accuracy	Overheads		
					Inf. Performance	Area	Power
TEEs and Attestation	Tramer et al. [114]	○	●	-0.5%	$2.5-12.5\times$	◐	18.3%
	Chen et al. [115]	●	○	○	$0.7-1.9\%$	◐	◐
	Chakraborty et al. [116]	●	○	○	○	◐	◐
	Lin et al. [117]	○	●	○	○	◐	◐
	Sun et al. [118]	○	●	1%	$0.6-1.6\times$	◐	◐
	Liu et al. [119]	○	●	○	53%	◐	◐
Hardware Masking	Dubey et al. [77]	○	●	◐	$2.8\times$	$2.3\times$	◐
	Dubey et al. [96]	○	●	◐	3.5%	$5.9\times$	◐
	Dubey et al. [97]	○	●	◐	3.1%	$5.3-5.7\times$	◐
	Dubey et al. [98]	○	●	$0.5-1\%$	$2.5-3.6\times$	3.1%	◐
	Breier et al. [120]	○	●	◐	$2.6-11\%$	◐	◐
	Dubey et al. [121]	○	●	◐	$2.1\times$	$\approx 3\times$	◐
	Maji et al. [122]	○	●	$< 5\%$	$1.4\times$	$1.64\times$	$5.5\times$
OOP Defenses	He et al. [123]	○	●	○	2%	◐	◐
	Zhou et al. [124]	○	●	◐	$0.14\times$	◐	◐
	Zhou et al. [125]	○	●	◐	◐	◐	◐
	Szentannai et al. [126]	○	●	○	◐	◐	◐
	DeepObfuscation [127]	○	●	○	○	$2.1\times$	◐
	Isakov et al. [128]	○	●	◐	◐	◐	Low Power
	Wang et al. [129]	○	●	◐	○	$< 0.001\%$	$< 0.01\%$
	Ahmadi et al. [130]	○	●	○	$1.4\times$	◐	◐
	Olney et al. [131]	○	●	○	4.4%	○	○
	Sternby et al. [132]	○	●	$< 1\%$	◐	◐	◐
	Zhang et al. [133]	○	●	○	○	1.17%	◐
	Jap et al. [134]	○	●	○	○	○	○
	Luo et al. [135]	○	●	◐	3.06%	◐	◐
	Duan et al. [136]	○	●	○	21%	◐	◐
Cryptographic	Zuo et al. [36]	○	●	◐	$\approx 5\%$	◐	◐
	Wei et al. [137]	○	●	2%	$0.77\mu s/weight$	◐	◐
	Alam et al. [138]	○	●	○	$2.5-3\times$	◐	◐
	Alam et al. [139]	○	●	○	$1.3-2\times$	Negligible	◐
	Zhou et al. [140]	○	●	◐	◐	◐	◐
	Wang et al. [141]	○	●	○	$1.03-1.25\times$	6.4%	9.6%
SDR Defenses	Liu et al. [142]	○	●	◐	Constant factor	◐	◐
	Ganesan et al. [143]	○	●	○	0.56%	2.46%	3.28%
Hardware-Assisted	Chakraborty et al. [144]	○	●	○	○	0.5%	◐
	Goldstein et al. [145]	○	●	○	◐	◐	◐
	Grailoo et al. [146]	○	●	○	○	$5-21\%$	$0.7-9\%$
	Rajasekharan [147]	○	●	◐	◐	◐	◐
	Mankali et al. [148]	○	●	◐	◐	◐	◐
	Li et al. [149]	○	●	○	Negligible	Negligible	Negligible
	Li et al. [150]	○	●	○	Negligible	Negligible	Negligible
	Zou et al. [151]	○	●	◐	$16\times$	$< 2\%$	◐
Prediction Poisoning	Grailoo et al. [152]	○	●	○	○	Negligible	Negligible
Fingerprinting	Cao et al. [153]	●	○	○	Negligible	◐	◐
	Shen et al. [154]	●	○	○	Negligible	◐	◐
Compression	Shrivastava et al. [155]	○	●	◐	$1.71\times$	Negligible	Negligible

● denotes "Yes", and ○ denotes "No", ✗ denotes "Not Applicable", ◐ denotes "Not mentioned". SDR stands for "Shuffling, Dummy Operation Insertion, and Randomization Defenses." OOP stands for Obfuscation, Optimization, and Perturbation Techniques.

10.5. SDR Defenses

This category contains techniques that shuffle instruction sequences during execution, memory locations to obfuscate the address space with low overheads, etc.; and techniques for dummy instruction insertion including insertion of dummy memory access requests; as well as techniques for address space layout randomization. This class of defenses aim at confusing the attacker in terms of hardware execution or memory patterns, with the objective to thwart the RE process.

10.6. Hardware-Assisted Defenses

This category considers technique that borrow techniques from hardware security used for IP protection and fingerprinting like logic-locking, physical unclonable functions (PUFs), etc. Researchers have borrowed these techniques from other fields and successfully applied them for model IP protection. These are typically supply chain defenses from the hardware security community, hence hardware masking do not fall in this category.

10.7. Prediction Poisoning Defenses

As discussed earlier, architecture stealing for distillation-based attacks has been very popular among hardware attacks. This category includes methods to poison the predictions that prevent such hardware based distillation-based attacks.

10.8. Fingerprinting Defenses

These defenses create a fingerprint of the original classifier, so as to distinguish from pirated classifiers. For example, one of the techniques creates a fingerprint of some data points near the classification boundary of the original classifier, to characterize its uniqueness [153].

10.9. Model Compression Defenses

This class of techniques use compression as a method of obfuscation and a potential source of randomness [155]. As shown in Table 2, except a few methods, most of the works have not quantified area and power overheads. This is a major limitation, and needs to be addressed for future defenses, since without quantifying that, it becomes difficult for chip designers to make design decisions without considering PPA requirements.

Recommendation: Future defenses need to quantify area and power/energy overheads.

Table 2 also shows that most of the defenses are prevention-based and with little impact on classifier accuracy, thus making them very attractive. However, except for a few defenses [115], [156], most of them have either very high performance overheads or area overheads. They are typically many times original design size. This is typically not acceptable for designers, hence is an important consideration to keep the overheads a small percentage of original design for future defenses.

Recommendation: Future defenses need to keep performance and area overheads to a small percentage of original design.

10.10. Scan-chain defenses

In the case of scan-chain attacks [23], [24], there are no defenses so far. To handle in-field errors e.g. silent data corruption (SDC), most of the semiconductor companies are actively looking at scan for in-field testing. As a result, the threat becomes even more relevant and important to defend.

Open Problem: Is it possible to defend scan-chain attacks for NN hardware accelerators?

There has been a significant body of work [113], [159], [160], [161] on scan-chain defenses for cryptographic implementations. However, there is no validation done to check if these defenses are effective/ineffective for NN hardware.

Open Problem: Are scan-chain defenses for cryptographic implementations effective/ineffective on NN hardware to defend against model RE attacks?

11. Models

The attack procedure depends on the type of the model-under-attack and the impact of the attack depends on the csot of the model. So far, researchers have been discussing the possibility of the private training data being under risk due to model RE. However, there is no significant work that performs the security evaluation pertaining to the same.

Open Problem: Is it possible to reverse engineer the private training data based on the reverse-engineered NN model through hardware?

Further, researchers have evaluated mostly on CNNs and MLPs. Bringing the threat model into perspective, the original goal was to protect the commercial value of the model IP. As a result, security evaluation and protection becomes important when the value of the model IP is very high. Neural networks like CNNs, MLPs are very important IPs because sometimes the cost of training and neural-architecture search (NAS) can be as high as $\$3$ million. However, in the case of some of the natural language processing (NLP) tasks, the cost goes up even more. For example, transformers, which are used in `GPT-4` cost more than $\$100$ million for training [6]. However, there is no work so far neither on the security evaluation side nor on the protection front.

Open Problem: Extending model RE to expensive IPs like transformers.

11.1. Adversarial Attacks

Although a significant number of papers focus on *architecture stealing* (refer Figure 6), and hint that the adversary can exploit it for adversarial attacks, few have actually explored that direction. One such work is `Tenet` [34], which implements and reports the adversarial success rate to be 42.4%. As a result, this is clearly an open research direction.

Open Problem: Can model RE through hardware help adversaries craft adversarial examples/samples?

12. Conclusion

Since deep learning gained popularity in the recent years [3], model reverse engineering problem has been well-studied from both software and hardware perspectives. Although there has been significant body of work (130+ papers) on the hardware side, there is no systematization of knowledge. We addressed this issue by categorizing the various attacks/defenses based on various criterion. We also discussed pros and cons of these works based on different figures of merit, and suggested several open research directions.

Acknowledgments

The authors would like to thank Manideep Thotakura from the University at Albany for assisting with collecting the papers.

References

[1] T. Nayan, Q. Guo, M. Al Duniawi, M. Botacin, S. Uluagac, and R. Sun, "SoK: All You Need to Know About On-Device ML Model Extraction-The Gap Between Research and Practice."

[2] D. Oliynyk, R. Mayer, and A. Rauber, "I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences," *ACM Comput. Surv.*, vol. 55, no. 14s, jul 2023. [Online]. Available: https://doi.org/10.1145/3595292

[3] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[4] L. Batina, S. Bhasin, D. Jap, and S. Picek, "SCA Strikes Back: Reverse-Engineering Neural Network Architectures Using Side Channels," *IEEE Design Test*, vol. 39, no. 4, pp. 7–14, 2022.

[5] "Model Ownership," 2024. [Online]. Available: https://sambanova.ai/blog/generative-ai-model-ownership

[6] "OpenAI's CEO Says the Age of Giant AI Models Is Already Over," 2023. [Online]. Available: https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/

[7] I. Mosafi, E. David, and N. S. Netanyahu, "Zero-Knowledge Attack for Replicating Protected Deep Neural Networks," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023, pp. 1–8.

[8] A. Dmitrenko, "DNN model extraction attacks using prediction interfaces," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:208977276

[9] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[10] C. Yang, L. Xie, S. Qiao, and A. L. Yuille, "Knowledge Distillation in Generations: More Tolerant Teachers Educate Better Students," *CoRR*, vol. abs/1805.05551, 2018. [Online]. Available: http://arxiv.org/abs/1805.05551

[11] J. R. C. da Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, "Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data," *CoRR*, vol. abs/1806.05476, 2018. [Online]. Available: http://arxiv.org/abs/1806.05476

[12] Y. Shi, Y. E. Sagduyu, K. Davaslioglu, and J. H. Li, "Active Deep Learning Attacks under Strict Rate Limitations for Online API Calls," *CoRR*, vol. abs/1811.01811, 2018. [Online]. Available: http://arxiv.org/abs/1811.01811

[13] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Model extraction and active learning," *CoRR*, vol. abs/1811.02054, 2018. [Online]. Available: http://arxiv.org/abs/1811.02054

[14] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. K. Shevade, and V. Ganapathy, "A framework for the extraction of Deep Neural Networks by leveraging public data," *CoRR*, vol. abs/1905.09165, 2019. [Online]. Available: http://arxiv.org/abs/1905.09165

[15] I. Mosafi, E. David, and N. S. Netanyahu, "DeepMimic: Mentor-Student Unlabeled Data Based Training," *CoRR*, vol. abs/1912.00079, 2019. [Online]. Available: http://arxiv.org/abs/1912.00079

[16] ——, "Stealing knowledge from protected deep neural networks using composite unlabeled data," *CoRR*, vol. abs/1912.03959, 2019. [Online]. Available: http://arxiv.org/abs/1912.03959

[17] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. K. Shevade, and V. Ganapathy, "ActiveThief: Model Extraction Using Active Learning and Unannotated Public Data," in *AAAI Conference on Artificial Intelligence*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:213157375

[18] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing Machine Learning Models via Prediction APIs," *CoRR*, vol. abs/1609.02943, 2016. [Online]. Available: http://arxiv.org/abs/1609.02943

[19] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High-Fidelity Extraction of Neural Network Models," *CoRR*, vol. abs/1909.01838, 2019. [Online]. Available: http://arxiv.org/abs/1909.01838

[20] S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt, "Model reconstruction from model explanations," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT* '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–9. [Online]. Available: https://doi.org/10.1145/3287560.3287562

[21] D. Rolnick and K. P. Körding, "Identifying Weights and Architectures of Unknown ReLU Networks," *CoRR*, vol. abs/1910.00744, 2019. [Online]. Available: http://arxiv.org/abs/1910.00744

[22] R. Ge, R. Kuditipudi, Z. Li, and X. Wang, "Learning two-layer neural networks with symmetric inputs," *CoRR*, vol. abs/1810.06793, 2018. [Online]. Available: http://arxiv.org/abs/1810.06793

[23] S. Potluri and A. Aysu, "Stealing Neural Network Models through the Scan Chain: A New Threat for ML Hardware," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–8.

[24] S. Jiang, S. Potluri, and T.-Y. Ho, "Scalable scan-chain-based extraction of neural network models," in *2023 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2023, pp. 1–6.

[25] T. Nayan, Q. Guo, M. Al Duniawi, M. Botacin, S. Uluagac, and R. Sun, "SoK: All You Need to Know About On-Device ML Model Extraction-The Gap Between Research and Practice."

[26] M. Isakov, V. Gadepally, K. M. Gettings, and M. A. Kinsy, "Survey of attacks and defenses on edge-deployed neural networks," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–8.

[27] Q. Xu, M. T. Arafin, and G. Qu, "Security of neural networks from hardware perspective: A survey and beyond," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. Association for Computing Machinery, 2021, p. 449–454.

[28] S. Mittal, H. Gupta, and S. Srivastava, "A survey on hardware security of DNN models and accelerators," *Journal of Systems Architecture*, vol. 117, p. 102163, 2021.

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

[30] F. Sultana, A. Sufian, and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," in *International Conference on Research in Computational Intelligence and Communication Networks*, 2018, pp. 122–129.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[33] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 515–532. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/batina

[34] C. Gao, B. Li, Y. Wang, W. Chen, and L. Zhang, "Tenet: A Neural Network Model Extraction Attack in Multi-core Architecture," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. Association for Computing Machinery, 2021, p. 21–26.

[35] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes Attack: Steal DNN Models with Lossless Inference Accuracy," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1973–1988. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/zhu

[36] P. Zuo, Y. Hua, L. Liang, X. Xie, X. Hu, and Y. Xie, "Sealing neural network models in encrypted deep learning accelerators," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1255–1260.

[37] L. Huegle, M. Gotthard, V. Meyers, J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Power2Picture: Using Generative CNNs for Input Recovery of Neural Network Accelerators through Power Side-Channels on FPGAs," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023, pp. 155–161.

[38] "Xilinx AI Engine." [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp506-ai-engine.pdf

[39] "NVIDIA AI Engine." [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-1/

[40] "IBM TrueNorth." [Online]. Available: https://www.research.ibm.com/artificial-intelligence/hardware/

[41] "ARM ML Processor." [Online]. Available: https://www.arm.com/solutions/artificial-intelligence

[42] "Intel Movidius." [Online]. Available: https://software.intel.com/en-us/movidius-ncs

[43] "Qualcomm AI Engine." [Online]. Available: https://www.qualcomm.com/snapdragon/artificial-intelligence

[44] "Nxp s32v234 vision processor." [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform/vision-processor-for-front-and-surround-view-camera-machine-learning-and-sensor-fusion:S32V234

[45] "Texas-instruments ai engine." [Online]. Available: http://www.ti.com/tool/SITARA-MACHINE-LEARNING

[46] "Samsung neural processing unit (npu)." [Online]. Available: https://www.samsung.com/global/galaxy/what-is/npu/

[47] "Google Tensor Processing Unit (TPU)." [Online]. Available: https://cloud.google.com/tpu/docs/tpus

[48] M. Jagielski et al, "High Accuracy and High Fidelity Extraction of Neural Networks, booktitle = USENIX Security Symposium, pages = 1345–1362, year = 2020,."

[49] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Appsat: Approximately deobfuscating integrated circuits," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 95–100.

[50] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitras, "Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks," *CoRR*, vol. abs/1810.03487, 2018. [Online]. Available: http://arxiv.org/abs/1810.03487

[51] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1299–1316.

[52] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[53] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels. corr abs/1812.11720 (2018)," *arXiv preprint arXiv:1812.11720*, 2018.

[54] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2003–2020. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/yan

[55] H. Jeong, D. Ryu, and J. Hur, "Neural Network Stealing via Meltdown," in *2021 International Conference on Information Networking (ICOIN)*, 2021, pp. 36–38.

[56] Y. Liu and A. Srivastava, "GANRED: GAN-based Reverse Engineering of DNNs via Cache Side-Channel," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2020, p. 41–52.

[57] S. Wolf, H. Hu, R. Cooley, and M. Borowczak, "Stealing machine learning parameters via side channel power attacks," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 242–247.

[58] 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," *arXiv preprint arXiv:2309.11894*, 2023.

[59] C. Gongye, Y. Fei, and T. Wahl, "Reverse-engineering deep neural networks using floating-point timing side-channels," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[60] S. Maji, U. Banerjee, and A. P. 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*, vol. 8, no. 15, pp. 12 079–12 092, 2021.

[61] E. Malan, V. Peluso, A. Calimera, and E. Macii, "Enabling DVFS Side-Channel Attacks for Neural Network Fingerprinting in Edge Inference Services," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2023, pp. 1–6.

[62] D. Ryu, Y. Kim, and J. Hur, "γ-Knife: Extracting Neural Network Architecture Through Software-Based Power Side-Channel," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–17, 2023.

[63] Y.-S. Won, S. Chatterjee, D. Jap, A. Basu, and S. Bhasin, "DeepFreeze: Cold Boot Attacks and High Fidelity Model Recovery on Commercial EdgeML Device," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[64] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "Leaky DNN: Stealing Deep-Learning Model Secret with GPU Context-Switching Side-Channel," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 125–137.

[65] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints," ser. ACM ASPLOS, 2020, p. 385–399.

[66] Ł. Chmielewski and L. Weissbart, "On Reverse Engineering Neural Network Implementation on GPU," in *Applied Cryptography and Network Security Workshops*. Cham: Springer International Publishing, 2021, pp. 96–113.

[67] 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 a Distance," in *IEEE Security and Privacy Workshops (SPW)*, 2022, pp. 312–322.

[68] 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," *arXiv preprint arXiv:2208.13720*, 2022.

[69] M. M. Ahmadi, L. Alrahis, A. Colucci, O. Sinanoglu, and M. Shafique, "NeuroUnlock: Unlocking the Architecture of Obfuscated Deep Neural Networks," in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 01–10.

[70] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. A. Faruque, "Stealing Neural Network Structure Through Remote FPGA Side-Channel Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[71] H. Yu et al, "DeepEM: Deep Neural Networks Model Recovery through EM Side-Channel Information Leakage," in *HOST*, 2020.

[72] 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 *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[73] A. Dubey, E. Karabulut, A. Awad, and A. Aysu, "High-fidelity model extraction attacks via remote power monitors," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2022, pp. 328–331.

[74] V. Meyers, D. Gnad, and M. Tahoori, "Reverse Engineering Neural Network Folding with Remote FPGA Power Analysis," in *IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–10.

[75] T. Li and C. Merkel, "Model extraction and adversarial attacks on neural networks using switching power information," in *Artificial Neural Networks and Machine Learning – ICANN 2021*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds. Cham: Springer International Publishing, 2021, pp. 91–101.

[76] 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 *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

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

[78] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[79] S. Sharma, U. Kamal, J. Tong, T. Krishna, and S. Mukhopadhyay, "SNATCH: Stealing Neural Network Architecture from ML Accelerator in Intelligent Sensors," in *2023 IEEE SENSORS*, 2023, pp. 1–4.

[80] J. Read, W. Li, and S. Yu, "A method for reverse engineering neural network parameters from compute-in-memory accelerators," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 302–307.

[81] Y.-S. Won, S. Chatterjee, D. Jap, S. Bhasin, and A. Basu, "Time to leak: Cross-device timing attack on edge deep learning accelerator," in *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, 2021, pp. 1–4.

[82] D. Yang, P. J. Nair, and M. Lis, "HuffDuff: Stealing Pruned DNNs from Sparse Accelerators," ser. ACM ASPLOS, 2023, p. 385–399.

[83] 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 (SP)*, 2024, pp. 5–5.

[84] X. Yan, X. Lou, G. Xu, H. Qiu, S. Guo, C. H. Chang, and T. Zhang, "MERCURY: An Automated Remote Side-channel Attack to Nvidia Deep Learning Accelerator," in *IEEE International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 188–197.

[85] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "Remote Power Attacks on the Versatile Tensor Accelerator in Multi-Tenant FPGAs," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 242–246.

[86] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[87] S. Chandrasekar, S.-K. Lam, and S. Thambipillai, "DNN Model Theft Through Trojan Side-Channel on Edge FPGA Accelerator," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, F. Palumbo, G. Keramidas, N. Voros, and P. C. Diniz, Eds. Cham: Springer Nature Switzerland, 2023, pp. 146–158.

[88] R. Wu, T. Kim, D. J. Tian, A. Bianchi, and D. Xu, "DnD: A Cross-Architecture Deep Neural Network Decompiler," in *USENIX Security Symposium*, 2022, pp. 2135–2152.

[89] B. Olney and R. Karam, "Bits to BNNs: Reconstructing FPGA ML-IP with Joint Bitstream and Side-Channel Analysis," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2023, pp. 238–248.

[90] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware trojans through side-channel engineering," in *CHES*. Springer, 2009, pp. 382–395.

[91] M. Ender, S. Ghandali, A. Moradi, and C. Paar, "The first thorough side-channel hardware trojan," in *ASIACRYPT*. Springer, 2017, pp. 755–780.

[92] J. Zhang, G. Su, Y. Liu, L. Wei, F. Yuan, G. Bai, and Q. Xu, "On Trojan side channel design and identification," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 278–285.

[93] T. Perez, M. Imran, P. Vaz, and S. Pagliarini, "Side-channel trojan insertion-a practical foundry-side attack via ECO," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[94] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO*. Springer, 1999, pp. 388–397.

[95] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and counter-measures for smart cards," in *Smart Card Programming and Security: International Conference on Research in Smart Cards, E-smart 2001 Cannes, France, September 19–21, 2001 Proceedings*. Springer, 2001, pp. 200–210.

[96] A. Dubey, R. Cammarota, and A. Aysu, "BoMaNet: Boolean Masking of an Entire Neural Network," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*. IEEE, 2020, pp. 51:1–51:9. [Online]. Available: https://doi.org/10.1145/3400302.3415649

[97] A. Dubey, R. Cammarota, V. Suresh, and A. Aysu, "Guarding machine learning hardware against physical side-channel attacks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–31, 2022.

[98] A. Dubey, A. Ahmad, M. A. Pasha, R. Cammarota, and A. Aysu, "ModuloNET: Neural Networks Meet Modular Arithmetic for Efficient Hardware Masking," *TCHES*, vol. 2022, no. 1, pp. 506–556, 2022. [Online]. Available: https://doi.org/10.46586/tches.v2022.i1.506-556

[99] A. Dubey and A. Aysu, "A Full-Stack Approach for Side-Channel Secure ML Hardware," in *2023 IEEE International Test Conference (ITC)*, 2023, pp. 186–195.

[100] D. J. Bernstein, "Cache-timing attacks on AES," 2005.

[101] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 229–244.

[102] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1157–1174.

[103] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[104] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 695–711.

[105] J. Breier, D. Jap, X. Hou, S. Bhasin, and Y. Liu, "SNIFF: reverse engineering of neural networks with fault attacks," *IEEE Transactions on Reliability*, vol. 71, no. 4, pp. 1527–1539, 2021.

[106] K. Hector, P.-A. Moëllic, J.-M. Dutertre, and M. Dumont, "Fault injection and safe-error attack for extraction of embedded neural network models," in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 644–664.

[107] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 131–138.

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

[109] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2019, pp. 48–55.

[110] P. Zhao, S. Wang, C. Gongye, Y. Wang, Y. Fei, and X. Lin, "Fault sneaking attack: A stealthy framework for misleading deep neural networkss," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[111] V. Meyers, D. Gnad, and M. Tahoori, "Active and passive physical attacks on neural network accelerators," *IEEE Design & Test*, 2023.

[112] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Physical security of deep learning on edge devices: Comprehensive evaluation of fault injection attack vectors," *Microelectronics Reliability*, vol. 120, p. 114116, 2021.

[113] B. Yang et al, "Scan based side channel attack on dedicated hardware implementations of Data Encryption Standard," in *ITC*, 2004, pp. 339–344.

[114] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[115] H. Chen, C. Fu, B. D. Rouhani, J. Zhao, and F. Koushanfar, "DeepAttest: an end-to-end attestation framework for deep neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*. Association for Computing Machinery, 2019, p. 487–498.

[116] A. Chakraborty, D. Xing, Y. Liu, and A. Srivastava, "Dynamarks: Defending against deep learning model extraction using dynamic watermarking," *arXiv preprint arXiv:2207.13321*, 2022.

[117] H.-Y. Lin, C. Fang, and J. Shi, "Bident Structure for Neural Network Model Protection," in *ICISSP*, 2020, pp. 377–384.

[118] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, "ShadowNet: A Secure and Efficient On-device Model Inference System for Convolutional Neural Networks," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1596–1612.

[119] Z. Liu, Y. Lu, X. Xie, Y. Fang, Z. Jian, and T. Li, "Trusted-DNN: A TrustZone-based Adaptive Isolation Strategy for Deep Neural Networks," in *Proceedings of the ACM Turing Award Celebration Conference - China*, ser. ACM TURC '21. Association for Computing Machinery, 2021, p. 67–71.

[120] J. Breier, D. Jap, X. Hou, and S. Bhasin, "A desynchronization-based countermeasure against side-channel analysis of neural networks," in *Cyber Security, Cryptology, and Machine Learning*. Springer Nature Switzerland, 2023, pp. 296–306.

[121] A. Dubey, R. Cammarota, A. Varna, R. Kumar, and A. Aysu, "Hardware-software co-design for side-channel protected neural network inference," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2023, pp. 155–166.

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

[123] J. Li, Z. He, A. S. Rakin, D. Fan, and C. Chakrabarti, "NeurObfuscator: A Full-stack Obfuscation Tool to Mitigate Neural Architecture Stealing," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 248–258.

[124] T. Zhou, S. Ren, and X. Xu, "ObfuNAS: A Neural Architecture Search-Based DNN Obfuscation Approach." New York, NY, USA: Association for Computing Machinery, 2022.

[125] T. Zhou, Y. Luo, S. Ren, and X. Xu, "NNSplitter: An Active Defense Solution for DNN Model via Automated Weight Obfuscation," in *International Conference on Machine Learning*, vol. 202. PMLR, 2023, pp. 42 614–42 624.

[126] K. Szentannai, J. Al-Afandi, and A. Horváth, "Preventing neural network weight stealing via network obfuscation," in *Intelligent Computing, Volume 3*. Springer, 2020, pp. 1–11.

[127] H. Xu, Y. Su, Z. Zhao, Y. Zhou, M. R. Lyu, and I. King, "DeepObfuscation: Securing the structure of convolutional neural networks via knowledge distillation," *arXiv preprint arXiv:1806.10313*, 2018.

[128] M. Isakov, L. Bu, H. Cheng, and M. A. Kinsy, "Preventing neural network model exfiltration in machine learning hardware accelerators," in *IEEE Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2018, pp. 62–67.

[129] Y. Wang, S. Jin, and T. Li, "A Low Cost Weight Obfuscation Scheme for Security Enhancement of ReRAM Based Neural Network Accelerators," in *ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2021, p. 499–504.

[130] M. M. Ahmadi, L. Alrahis, O. Sinanoglu, and M. Shafique, "DNN-Alias: Deep Neural Network Protection Against Side-Channel Attacks via Layer Balancing," *arXiv preprint arXiv:2303.06746*, 2023.

[131] B. Olney and R. Karam, "Protecting Deep Neural Network Intellectual Property with Architecture-Agnostic Input Obfuscation," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2022, p. 111–115.

[132] J. Sternby, B. Johansson, and M. Liljenstam, "Neural Network Model Obfuscation through Adversarial Training," in *IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 782–789.

[133] J. Zhang, C. Wang, Y. Cai, Z. Zhu, D. Kline, H. Yang, and Y. Wang, "WESCO: Weight-encoded Reliability and Security Co-design for In-memory Computing Systems," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 296–301.

[134] D. Jap and S. Bhasin, "Using Model Optimization as Countermeasure against Model Recovery Attacks," in *International Conference on Applied Cryptography and Network Security*. Springer, 2023, pp. 196–209.

[135] Y. Luo, S. Duan, C. Gongye, Y. Fei, and X. Xu, "NNReArch: A Tensor Program Scheduling Framework Against Neural Network Architecture Reverse Engineering," in *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–9.

[136] S. Duan, S. Ren, and X. Xu, "HDLock: Exploiting Privileged Encoding to Protect Hyperdimensional Computing Models against IP Stealing," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 679–684.

[137] W. Jiang, Z. Song, J. Zhan, D. Liu, and J. Wan, “Layerwise security protection for deep neural networks in industrial cyber physical systems,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 8797–8806, 2022.

[138] M. Alam, S. Saha, D. Mukhopadhyay, and S. Kundu, “Deep-Lock: Secure Authorization for Deep Neural Networks,” *CoRR*, vol. abs/2008.05966, 2020. [Online]. Available: https://arxiv.org/abs/2008.05966

[139] ——, “NN-Lock: A Lightweight Authorization to Prevent IP Threats of Deep Learning Models,” *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, apr 2022.

[140] J. Zhou and X. Zhang, “Joint protection scheme for deep neural network hardware accelerators and models,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 4518–4527, 2023.

[141] X. Wang, R. Hou, Y. Zhu, J. Zhang, and D. Meng, “NPUFort: A secure architecture of DNN accelerator against model inversion attack,” in *ACM International Conference on Computing Frontiers*, 2019, pp. 190–196.

[142] Y. Liu, D. Dachman-Soled, and A. Srivastava, “Mitigating reverse engineering attacks on deep neural networks,” in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 657–662.

[143] K. Ganesan, M. Fishkin, O. Lin, and N. E. Jerger, “BlackJack: Secure machine learning on IoT devices through hardware-based shuffling,” *arXiv preprint arXiv:2310.17804*, 2023.

[144] A. Chakraborty, A. Mondai, and A. Srivastava, “Hardware-assisted intellectual property protection of deep learning models,” in *ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[145] B. F. Goldstein, V. C. Patil, V. C. Ferreira, A. S. Nery, F. M. G. França, and S. Kundu, “Preventing DNN Model IP Theft via Hardware Obfuscation,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 267–277, 2021.

[146] M. Grailoo, U. Reinsalu, M. Leier, and T. Nikoubin, “Hardware-assisted Neural Network IP Protection using Non-malicious Backdoor and Selective Weight Obfuscation,” in *IEEE Dallas Circuit And System Conference (DCAS)*, 2022, pp. 1–6.

[147] D. Rajasekharan, N. Rangarajan, S. Patnaik, O. Sinanoglu, and Y. S. Chauhan, “SCANet: Securing the Weights With Superparamagnetic-MTJ Crossbar Array Networkss,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 9, pp. 5693–5707, 2023.

[148] L. Mankali, N. Rangarajan, S. Chatterjee, S. Kumar, Y. S. Chauhan, O. Sinanoglu, and H. Amrouch, “Leveraging Ferroelectric Stochasticity and In-Memory Computing for DNN IP Obfuscation,” *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 8, no. 2, pp. 102–110, 2022.

[149] W. Li, Y. Wang, H. Li, and X. Li, “P3M: a PIM-based neural network model protection scheme for deep learning accelerator,” in *ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019, p. 633–638.

[150] D. Li, D. Liu, Y. Guo, Y. Ren, J. Su, and J. Liu, “Defending against model extraction attacks with physical unclonable function,” *Information Sciences*, vol. 628, pp. 196–207, 2023.

[151] M. Zou, Z. Zhu, Y. Cai, J. Zhou, C. Wang, and Y. Wang, “Security Enhancement for RRAM Computing System through Obfuscating Crossbar Row Connections,” in *ACM/IEEE Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 466–471.

[152] M. Grailoo, Z. U. Abideen, M. Leier, and S. Pagliarini, “Preventing distillation-based attacks on neural network IP,” *CoRR*, vol. abs/2204.00292, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.00292

[153] X. Cao, J. Jia, and N. Z. Gong, “IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary,” ser. ACM AsiaCCS, 2021, p. 14–25.

[154] T. Shen, J. Qi, J. Jiang, X. Wang, S. Wen, X. Chen, S. Zhao, S. Wang, L. Chen, X. Luo, F. Zhang, and H. Cui, “SOTER: Guarding Black-box Inference for General Neural Networks at the Edge,” in *USENIX Annual Technical Conference (ATC)*, 2022, pp. 723–738.

[155] N. Shrivastava and S. R. Sarangi, “SparseLock: Securing Neural Network Models in Deep Learning Accelerators,” *arXiv preprint arXiv:2311.02628*, 2023.

[156] H. Chen, C. Fu, B. D. Rouhani, J. Zhao, and F. Koushanfar, “Intellectual Property Protection of Deep-Learning Systems via Hardware/Software Co-Design,” *IEEE Des. Test*, vol. 41, no. 2, pp. 23–31, 2024.

[157] M. Zhou, X. Gao, J. Wu, J. C. Grundy, X. Chen, C. Chen, and L. Li, “Model obfuscation for securing deployed neural networks,” 2022.

[158] M. Grailoo, Z. U. Abideen, M. Leier, and S. Pagliarini, “Preventing Distillation-based Attacks on Neural Network IP,” *arXiv preprint arXiv:2204.00292*, 2022.

[159] B. Yang, K. Wu, and R. Karri, “Secure scan: a design-for-test architecture for crypto chips,” in *ACM Annual Design Automation Conference (DAC)*, 2005, p. 135–140.

[160] S. S. Ali, S. M. Saeed, O. Sinanoglu, and R. Karri, “Novel test-mode-only scan attack and countermeasure for compression-based scan architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 808–821, 2015.

[161] Y. Liu, K. Wu, and R. Karri, “Scan-based attacks on linear feedback shift register based stream ciphers,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 2, 2011.