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.

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 (GPUs), (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.

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 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 parallelism.

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].

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
approximate model recovery methods exist, however analysis given a timeout. Although as discussed earlier, hard instances of the problem by performing incremental problem. Such techniques trade off query complexity some partial information, that helps him to perform stealing. It is because even for hard instances of the model RE an adversary is more powerful than an exact adversary. note that such security evaluation is critical, since such are missing in the field of model RE. It is important to
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-fidelity attacks. In the case of high-accuracy attacks, the adversary’s goal is to minimize \( \max |f(x) - \hat{f}(x)|, \forall x \in T \), where \( T \) is the dataset used to train the target model. On the other hand, in the case of high-fidelity 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 make 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 categorized based on different criteria, 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 to the background noise, thus other styles of attack are preferred. As discussed earlier, we categorize 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.

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 is 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 categorized 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 analyze 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 cryptosystems. 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

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

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 exploited 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 shared 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.

6.1.4. 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 possibilities, that need exploration.

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.

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.

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

---

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

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?
on NN hardware for model RE can be broadly categorized 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 the case with the exact model RE problem, because of 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 $256 \times 256$ 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 whatsoever 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) radiation 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 → 1, 1 → 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.

$$w_{n_1}^T x > -1 \times b_{n_1}$$
$$w_j^T x \leq -1 \times b_j, j \neq n_1$$

Figure 7. Linear Constraint Satisfaction (LCS) system for a target neuron-$n_1$ in the first layer of a fully connected network, where $x$ is the layer input, while $w_{n_1}^T$ 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.
9. Categorization Criteria for Defenses

Similar to attacks, the defenses can be categorized based on different criterions, 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 three categories: (a) trusted execution environments (TEE s) 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 $2 \times$ 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.

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.

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

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

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

11. Models

The attack procedure depends on the type of the model-under-attack and the impact of the attack depends on the cost 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.

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.

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


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.


