CLAIRVOYANCE: Exploiting Far-field EM Emanations of GPU to "See" Your DNN Models through Obstacles at a Distance

Sisheng Liang
Clemson University
Clemson, SC, USA
sishenl@clemson.edu

Zihao Zhan
University of Florida
Gainesville, FL, USA
zhan.zihao@ufl.edu

Fan Yao
University of Central Florida
Orlando, FL, USA
fan.yao@ucf.edu

Long Cheng
Clemson University
Clemson, SC, USA
lcheng2@clemson.edu

Zhenkai Zhang
Clemson University
Clemson, SC, USA
zhenkai@clemson.edu

2022 IEEE Security and Privacy Workshops (SPW) | 978-1-6654-9643-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SPW54247.2022.9833894

***Abstract*—Deep neural networks (DNNs) are becoming increasingly popular in real-world applications, and they are considered valuable assets of enterprises. In recent years, a number of model extraction attacks have been formulated that can be mounted to successfully steal proprietary DNN models. Nevertheless, previous model extraction attacks require either logical access to the target models or physical access to the victim machines, and thus are not suitable for performing model stealing in scenarios where an outside attacker is in the proximity but at a distance.**

In this paper, we propose a new model extraction attack named CLAIRVOYANCE that exploits certain far-field electromagnetic signals emanated from a GPU to steal DNN models at a distance of several meters away from the victim machine even with some obstacles in-between. Using CLAIRVOYANCE, an attacker can effectively deduce DNN architectures (e.g., the number of layers and their types) and layer configurations (e.g., the number of kernels, sizes of layers, and sizes of strides). We use several case studies (*e.g.*, VGG and ResNet) to demonstrate its effectiveness.

I. INTRODUCTION

In recent years, deep neural networks (DNNs) have been widely used for many artificial intelligence applications, such as computer vision [14], natural language processing [25], speech recognition [9], autonomous driving [21], [4], and gaming [23]. Given their increasingly high design cost, DNN models are often considered valuable pieces of intellectual property (IP) [37]. Naturally, they become the target of attackers who attempt to steal these models for certain illicit purposes (e.g., establishing plagiarized AI services [35]).

Recent advances in model extraction attacks have clearly demonstrated the ability of attackers in stealing DNN models. These attacks mainly fall into two categories: *learning-based approaches* [19], [20] and *side-channel-based approaches* [31], [16], [12], [28], [13], [32], [2], [1], [37]. The learning-based approach usually requires an attacker to interact with the victim model and related datasets to deduce the architecture and parameters of the model [19], [20]. Apparently, this type of methods needs the attacker to have logical access to the model (e.g., model query). The second class of model extraction attacks exploits various sources of side-channel information including signals both in the digital (e.g., microarchitecture attacks [31], [16], [12], [28], [13]) and physical domains (e.g., [32], [2], [1]) to infer DNN model secrets. Note that to exploit the logical side channels such as cache or bus contention, adversaries need to have logical access to the victim model's machine and share the hardware resources. When the attacker does not have logical access to the victim machine but is in its proximity, he/she may leverage physical side-channel information to achieve model extraction.

In fact, the state-of-the-art physical side-channel attacks on stealing DNNs are mainly based on electromagnetic (EM) signals. However, all the existing works exploit near-field EM signals to extract DNN models. To measure such signals, the attacker has to get very close to the victim devices [32], [1] (e.g., the sensor has to be mounted to the cable of GPU power supply in [1]), and sometimes even needs to decapsulate the targeted chip packages [2]. These methods are not applicable in common scenarios where the victim machines are physically isolated (e.g., wall-gapped) and are located several meters away from the attacker's reach. We note that a stealthy and realistic DNN model extraction attack by *exploiting physical side channels at a distance* has not been demonstrated.

In this paper, we propose CLAIRVOYANCE, a strong and realistic far-field EM-based model extraction attack, allowing the attacker to extract DNN model information at a distance of several meters away from the victim machine even with an obstacle (e.g., a wall) in-between. Specifically, we leverage the EM emanations from the GPU memory clock, which can propagate very far and contain enough information to reconstruct the DNN models. To the best of our knowledge, our work serves as the first far-field EM-based model extraction attack that steals DNN models running on a modern GPU at a distance.

We make the following contributions in this work:

1) We propose the first realistic DNN model extraction attack that can be mounted at a distance of several meters away from the victim machine equipped with a GPU. The long-range contactless exploitation poses a great threat as it can steal the DNN models much more stealthily than existing approaches.
2) We formulate a signal processing technique that can greatly increase the signal-to-noise ratio (SNR) to reveal hidden features of our interest in noisy measurements. This technique is necessary when the distance between the attacker and victim is large or they are separated by obstacles such as thick concrete walls.
3) We demonstrate successful attacks by extracting the architecture of DNN models (including the number of layers and the types of layers) and also inferring the specific configurations of each layer (including the number of kernels, the sizes of kernels, and the sizes

DOI 10.1109/SPW54247.2022.00036

of strides) based on the statistical analysis of the layer length.

II. Background

In this section, we first briefly describe the DNN models, and then we present basics about architecture of discrete GPUs that are widely used nowadays for deep learning systems. We also provide background information on the EM side-channel.

A. Deep Neural Networks

A deep neural network (DNN) is a computational model composed of multiple processing layers to learn abstract representations of data. The characteristics of a DNN are determined by two major aspects: (1) network architecture including the layer width (the number of units or the number of kernels, kernel size, stride, padding), layer depth, layer types, and connection topology between layers; (2) model parameters including weights, biases, and batch normalization parameters [32], [12].

The most commonly used structures of DNN are the chained structures and the length of the chain gives the depth of the model. With the chained structures, there are various ways to connect a pair of layers. For example, in the default forward neural networks or fully connected networks, every input unit is connected to every output unit. In convolutional networks that are designed for processing grid data (image) or time-serials data, layers are sparsely connected [7]. Usually, different types of layers are put together to accomplish a certain task together. For example, all of the popular DNN models for image processing such as AlexNet [14], VGGNet [24], ResNet[11], Inception-v3 [27], Inception-v4 [26] contain convolutional layers, fully connected layers, and pooling layers. However, they have different characteristics and performances because of different architecture designs.

B. GPU Architecture

Modern GPUs consist of a number of streaming multiprocessors (SMs), each containing a number of streaming processor (SP) cores, a multithreaded instruction fetch and issue unit and private/shared caches [5]. In each SM, multiple threads are grouped into a warp, which is scheduled by a warp manager. These threads share an instruction stream and will be executed simultaneously.

To increase the performance of handling large data, a GPU is often equipped with a large amount of DRAM that is shared among all the SMs [18]. This GPU memory system is managed independently from the host memory system and the data is transferred between the host memory and GPU memory via the PCIe bus [34]. There are multiple memory controllers which enables parallel access to the memory to achieve high bandwidth and high throughput [30].

C. Electromagnetic Emanations

Different computation activities induce distinct dynamic current changes in circuits. The current changes further lead to electromagnetic (EM) emanations. Therefore, the EM emanations carry the information about the computation activities. Such information has been exploited to mount realistic attacks [6], [15], [34], [33] as well as to help build effective defenses [36], [8], [17].

The EM signals are distributed widely across the spectrum. Periodic EM signals are easily to identify. For example, the EM signals created by periodic circuit activities like clocking and DRAM refreshing. Such EM signals are strong and can propagate very far. Moreover, some signals such as the DRAM clock signal are unintentionally modulated by memory access activities in the form of amplitude modulation (AM) [3].

III. Threat Model

In this paper, we assume that an attacker intends to steal the DNN model running on a victim machine. The victim machine is equipped with a modern NVIDIA GPU on which the target DNN model is executed. Although the attacker is assumed to be in the physical proximity of the victim machine, they may be separated by several meters. In addition, there may be some obstacles between the attacker and the victim machine (e.g., a wall). Under such a threat model, the attacker and the victim machine may be in different office rooms or cubicles, which makes our model extraction attack very stealthy and practical. Note that this threat model is much less restrictive and more realistic than the ones in prior works that assume the attacker has direct contact to victim machine's hardware (e.g., GPU power cable [1]).

Even though the attacker does not have any physical access to the victim machine, we assume that the attacker can find himself/herself an identical or similar GPU to the one used by the victim. The attacker uses this personal GPU for profiling. Moreover, the DNN model may contain various types of layers such as convolutional layer, fully connected layer, add layer, pooling layer and activation layer.

IV. EM Signals of Our Interest

The above-mentioned threat model arguably advocates an exploitation of certain EM side-channel information. In this section, we present an overview on the EM signals that we exploit for stealing the DNN model running on a modern GPU.

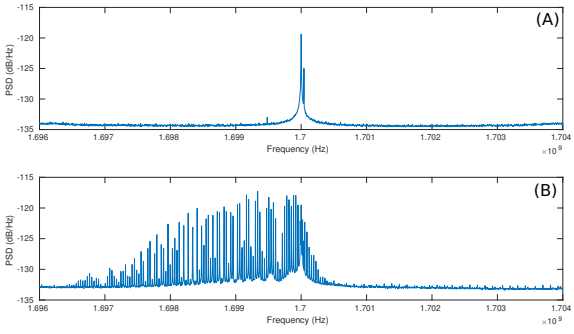


Fig. 1. Spectra of NVIDIA RTX 2080's EM signals at around 1700 MHz. (A) when the system is idle, and (B) when the GPU is running a DNN.

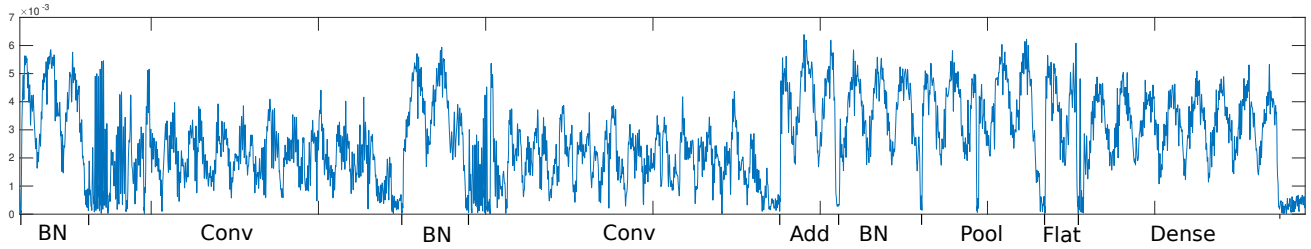


Fig. 2. A trace of EM signals at 1700 MHz when an NVIDIA RTX 2080 is running a DNN. BN: Batch Normalization layer, Conv: convolutional layer; Add: add layer; Pool: average pooling layer; Flat: flatten layer; Dense: fully connected layer. Horizontal axis represents time.

In [34], Zhan *et al.* observed some interesting phenomena. Basically, they found that the performance level of a modern GPU usually changes rapidly to seek a balance between performance and power consumption, and when a performance level is on/off, they observed the appearance/disappearance of clear EM signals at the corresponding GPU memory clock frequencies in the spectrum. (They further showed that such behavior can be exploited for inferring sensitive information.) More interestingly, they observed that for an NVIDIA GPU, if its performance level is fixed, there appears strong EM emanations that are inadvertently AM-modulated by the GPU memory accesses. The EM emanations are around the frequency that is one eighth the data rate in the cases of all NVIDIA GPUs. We verified their finding and will use such EM emanations to extract DNN models in this paper.

We notice that when an NVIDIA GPU runs a DNN model, its performance level will be bumped to a high level (usually the second highest) and stay there. Therefore, the EM signals of our interest will be at one eighth the memory transfer rate of that performance level. For example, given an NVIDIA RTX 2080 GPU, we find that when a DNN model starts running, its performance level will stay at level 3[1] whose memory transfer rate is 13600 MHz, and thus the EM signals of interest should be at 1700 MHz. Figure 1 illustrates this example. Figure 1 (A) shows the spectrum when the GPU is idle and Figure 1 (B) shows the spectrum when the a DNN model is running. We can see that there are noticeable signals in the frequency domain around 1700 MHz when the GPU runs the DNN model.

Note that there are many clear spectral components around 1700 MHz in Figure 1 (B), instead of just a single one. This phenomenon is due to a feature named spread spectrum clocking (SSC) [22], which varies the the clock frequency in a range so that the time spent by the clock signal at a particular frequency is reduced and the energy is spread over that range of frequencies [10]. The SSC technique is used for meeting the electromagnetic compatibility (EMC) standards.

Since different types of layers in a DNN model are implemented by different GPU kernels [1] and each kernel has its own way of accessing data, we speculate that executing a layer can potentially induce a distinct pattern of memory access activities. The pattern is determined by the type of layer, configurations of the layer, and also input shape of data. Given the fact that the EM emanations of our interest are AM-modulated by memory access activities [34], we should be able to infer some of the DNN model details from the EM traces.

To verify this hypothesis, we performed several experiments and Figure 2 shows an example. From the example, we can see that different types of layers induce distinct features in the trace which can be utilized to identify the types of layers. For instance, a batch normalization layer contains tall spikes with a short time interval; a convolutional layer starts with intense oscillations and ends with a gap; the outer envelope of a convolutional layer is similar to an arc; the trace of a dense layer is usually evenly distributed which is different from a trace of convolutional layer. Even though the length, height of the layer traces are affected by the network hyperparameters such as number of kernels, kernel size, stride, and input size, one can still recognize the type of layers by some relatively stable features such as the features of a convolutional layer described above.

V. Signal Enhancement

Despite the fact that we can infer DNN layer types from the EM emanations of our interest, these signals can become hard to exploit when the attacker is far away from the victim machine and/or when there are some physical obstacles like walls in-between. In such situations, the signal-to-noise ratio (SNR) can be significantly lowered such that the patterns used to infer DNN layer types may become unrecognizable. Therefore, we need to apply some signal processing techniques to enhance the signal.

Considering that we mainly leverage low-frequency components in the signal to identify different DNN layers, using a low-pass filter to remove high-frequency noise is seemingly effective to improve the SNR. However, designing such a low-pass filter is more challenging than it may appear. Since the SSC disperses the clock signal energy in a wide frequency range, as shown in Figure 1 (B), if the cutoff frequency is too low, some useful low-frequency features will be filtered out. On the other hand, if the cutoff frequency is too high, the low-pass filter will fail to remove all high-frequency noise effectively. This dilemma can not be solved by simply selecting an appropriate cutoff frequency. To solve this problem,

[1]NVIDIA RTX 2080 has five performance levels (0, 1, 2, 3, and 4). Level 3 is its second highest performance level.

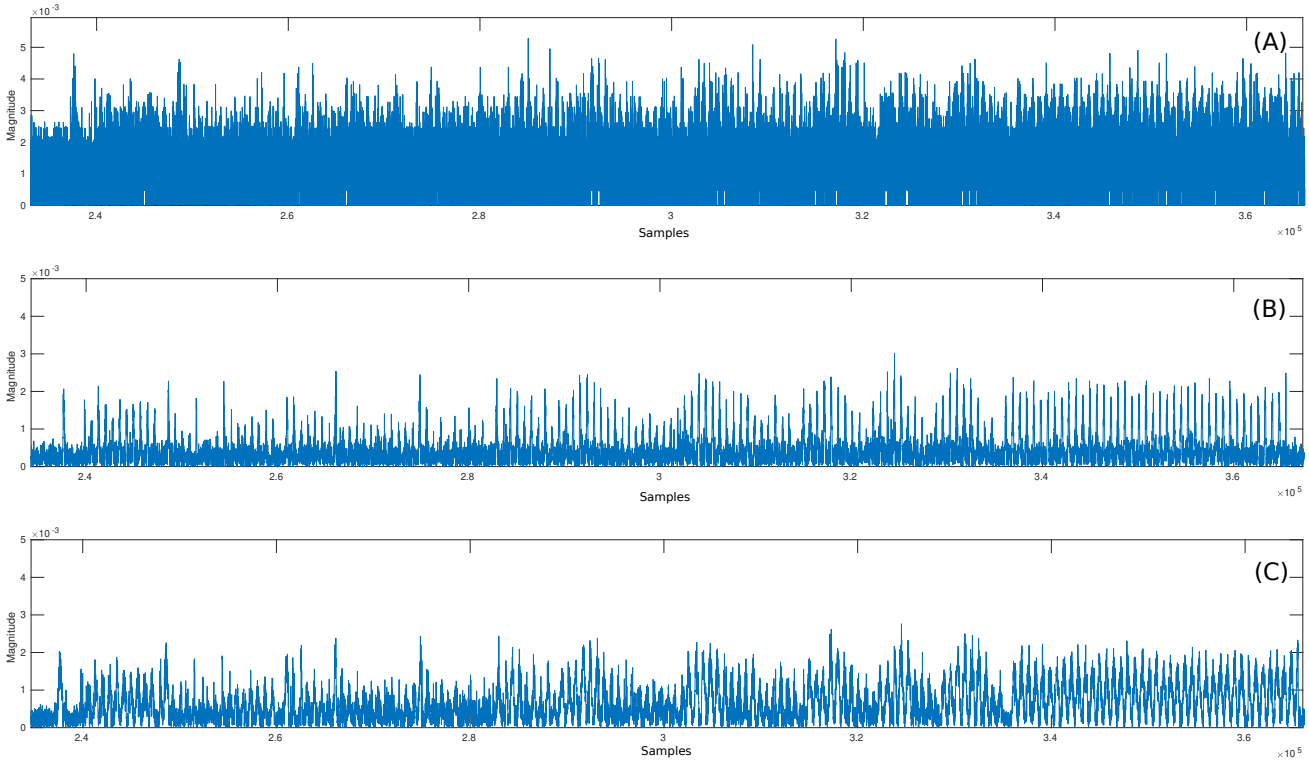


Fig. 3. Traces of EM signal of interest measured at a distance of 2 meters away from the victim machine with a wall in-between. (A) the original trace; (B) the same trace after direct low-pass filtering; (C) the same trace after de-spreading and low-pass filtering.

we combine the de-spreading technique proposed in [36] and a low-pass filter.

A. SSC De-spreading

We apply the de-spreading technique proposed in [36] to gather the scattered energy back together to facilitate noise reduction. The main idea is summarized in the following.

Given a clock signal whose frequency is f_c, SSC uses FM-modulation to vary the clock frequency in accordance with a signal $f_m(t)$ that is generated in the SSC hardware chip but undocumented. Normally, $f_m(t)$ is a periodic function, namely we have $f_m(t) = f_m(t + T_m)$ where T_m is the fundamental period of $f_m(t)$. At time t, the instantaneous frequency $f_i(t)$ of the clock signal becomes:

$$f_i(t) = f_c + K f_m(t) \ , \tag{1}$$

where K is some proportionality constant. In an analytic form, the effect of SSC is equivalent to multiplying the clock signal by a complex exponential function $\theta(t)$, which is defined as:

$$\theta(t) = e^{j2\pi \int_0^t K f_m(t) dt} \ , \tag{2}$$

where j denotes $\sqrt{-1}$. Hence, for the purpose of de-spreading, we just need to estimate $\theta(t)$ and multiply the measured signal by $\theta^{-1}(t)$.

The de-spreading technique presented in [36] can be summarized as the following steps:

1) Collect a sequence of samples at the clock frequency f_c. (In our case, the attacker can leverage the same GPU of his/her own and use the driver to set the performance level for collecting this sequence.)
2) Compute a phase difference sequence, i.e., $\delta_l = \phi_{l+1} - \phi_l$, where ϕ_l is the phase angle of the lth sample.
3) Find the fundamental period T_m of $f_m(t)$ by observing the repeated patterns in the phase difference sequence.
4) Derive a smaller sequence $\Delta = \{\delta_1, \delta_2, ...\}$ over one T_m. (To reduce noise, average multiple ones to obtain Δ.)
5) Align Δ with the targeted EM signals by cross-correlation.
6) Multiply each sample by $e^{-j\Phi}$, where Φ is the running sum of the aligned elements in Δ, to perform de-spreading.

For example, Figure 4 shows the spectrum of the EM signal of interest after de-spreading when running a DNN model on an NVIDIA RTX 2080 GPU. Compared with Figure 1 (B) that shows the spectrum without de-spreading, we can see that the energy of the signal is now more concentrated in a narrow frequency from 1699.8 MHz to 1700.2 MHz.

B. Low-Pass Filtering

To improve the signal's SNR, we remove the high-frequency noise using a low-pass filter. We select the appropriate cutoff

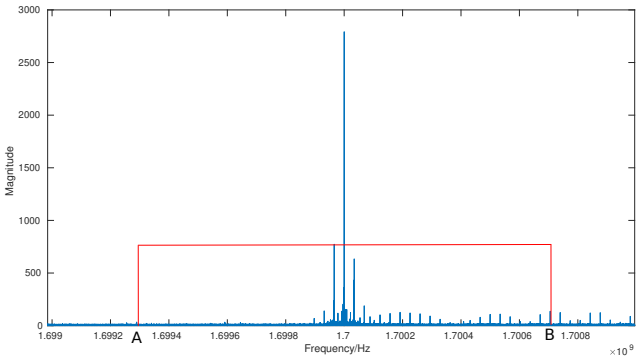


Fig. 4. Spectrum of the EM signal of our interest after de-spreading

frequency, as shown in Figure 4. Empirically, we find that keeping the frequency components in the band between 1699.3 MHz to 1700.7 MHz as shown in 4 and removing the frequency beyond this range gives a clear time series trace without losing the important features.

Figure 3 (A) shows a trace of the EM signal of our interest that is captured at a distance of 2 meters away from the victim machine with an office wall in-between. From this figure, one can barely recognize the DNN layers and their types because the detailed features are concealed by high-frequency noise. Figure 3 (B) shows the same trace after applying a low-pass filter directly without performing de-spreading first. While it is clearer, some important low-frequency features are actually lost. For example, there is a convolutional layer between the 240000^{th} and 244000^{th} samples. In general, to identify a convolutional layer, we should search for features such as an arc shape envelop and oscillations at the layer boundaries. However, these critical features are lost in trace (B) and can hardly be recognized from trace (A) due to noise. On the contrary, after de-spreading and filtering, we can see that most of the important layer features are preserved in Figure 3 (C) and also high-frequency noise is removed at the same time. From this example, one can find the effectiveness of our proposed signal processing techniques.

VI. Model Extraction

We extract the DNN model from the captured EM traces in two steps – step 1 is to recover the structure, and step 2 is to estimate each layer's configuration including the number of kernels, kernel size, and stride.

A. Topology Recovery

We recover the topology of a DNN by identifying the types of layers and the number of layers. We find that different layer types can induce different features in the captured EM traces, which makes our topology recovery possible. Although different layer configurations and/or input sizes can change the lengths and heights of corresponding trace segments, we notice that the characteristic features remain the same. For example, we observe that trace segments corresponding to convolutional layers with different numbers of kernels or strides have similar shapes. As shown in Figure 2, we can find two convolutional layers by identifying the oval shape envelope with oscillation at the beginning and a big gap at the end of the trace.

The execution time of a layer can also give us some hints about the layer type, especially helping us decide whether it is a weighted layer or not. Usually, a weighted layer (a convolutional layer or a fully connected layer in feedforward networks) has a longer execution time than a pooling layer or an add layer with similar input sizes. In Figure 2, we can also locate the dense layer combining the shape and the length of the trace segment.

Before launching the attack, the attacker can learn the EM trace features corresponding to different layer types on his/her own machine that is installed with the same type of GPU as the victim machine.

B. Layer Configuration Estimation

Aside from layer types, we need to recover the configuration of convolutional layers as well, including kernel sizes, the number of kernels, and strides. We find that such information can be estimated through the length of a trace segment corresponding to a layer.

We use $C = (k, s, n)$ to denote a specific configuration, where k denotes the kernel size, s denotes the stride size, and n is the number of kernels. Theoretically, the configuration search space is infinite, because each of the three elements has infinite possible values. However, there are actually a few possible values for each element of P in real-world DNN models. Table I lists the kernel sizes and strides used in the most successful CNNs. We can see that kernel sizes 3×3, 5×5, 7×7 are the most commonly used kernels size in popular CNNs. Moreover, at design time, a layer with a large kernel size is often replaced by multi-layers with smaller kernel sizes, which can reduce computation cost. For example, a 5×5 kernel size layer can be replaced by a two-layer network with 3×3 kernel size for cost saving purpose [27]. We can also see that stride 1 and 2 are the most commonly used ones. There are layers in some DNNs using stride 4, but we do not see designs with strides larger than that. We also find that 32, 64, 128, 256, and 512 are the most frequently used kernel numbers. Therefore, C actually has a very limited search space if we only consider reasonable configuration values.

TABLE I
The kernel sizes (k) / strides (s) in popular CNNs

AlexNet	VGG	ResNet	Inception-v3	Inception-v4
$11 \times 11/4$	$3 \times 3/1$	$7 \times 7/2$	$3 \times 3/1$	$3 \times 3/2$
$5 \times 5/1$		$3 \times 3/1$	$3 \times 3/2$	$3 \times 3/1$
$3 \times 3/1$		$3 \times 3/2$		

With a limited number of elements in the search space of C, the input sizes are also commonly limited to a few possibilities such as 28×28, 64×64, 96×96, 224×224, and 299×299. We find that the input fed to a DNN is often downsized even if its original size is much bigger.

We have conducted the statistics on the lengths of EM trace segments corresponding to layers with different configurations. We observe that, given a possible input size from the list aforementioned and a possible configuration C, the measured length of the EM trace segment has a certain distribution. There are two relationships between the trace segment length distributions. In the first one, there is considerable overlap between two length distributions. For example, Figure 5 shows the length distributions of two trace segments corresponding to two convolutional layers with different configurations. In this case, given a single trace segment length, it is hard to recognize directly which configuration it should be mapped to. However, we can capture the EM traces for multiple DNN runs to try to find the most plausible estimation following the likelihood principle. In the second relationship, there is barely any overlap between two length distributions. In such a case, we can directly recognize which configuration a layer should have.

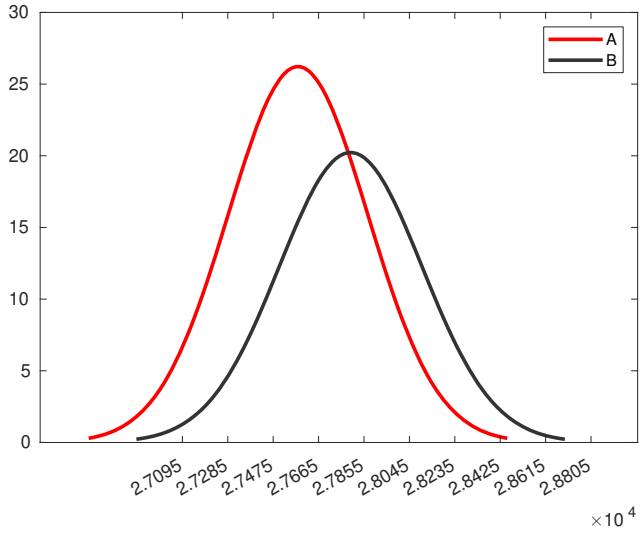


Fig. 5. Trace segment length distributions, where $C_A = (3, 1, 64)$ and $C_B = (5, 4, 256)$.

VII. Evaluation

This paper documents our preliminary work. In this section, we show some of our preliminary results.

A. Data Collection

We use a USRP B210 software defined radio (SDR) and a RFSPACE TSA900 antenna to collect the EM traces. We use the GNU Radio to manage the entire measurement process. The center frequency of the SDR is set to one eighth the memory transfer rate of the performance level at which DNN is running (e.g., 1700 MHz in terms of MSI RTX 2080). Firstly, the antenna is placed 2 meters away from the victim machine with no obstacles in-between. Secondly, the antenna is placed 2 meters away from the victim machine with a 16 cm wall (made of drywall) in-between as shown in Figure 6.

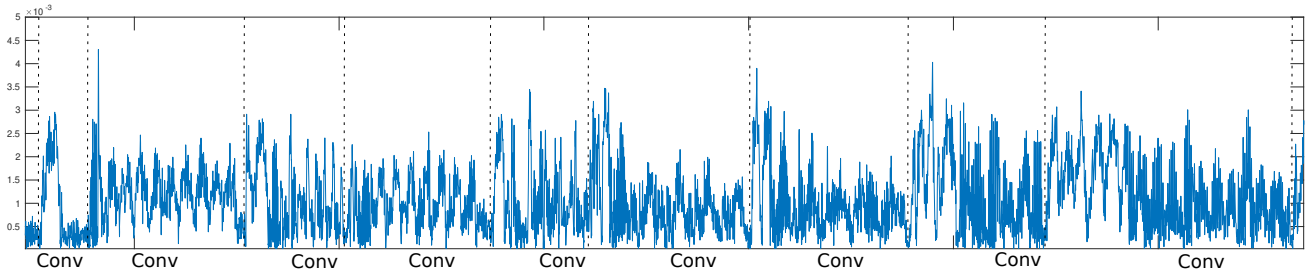
Fig. 6. Setup for the wall-penetrating model extraction attack.

B. DNN Layers Identification

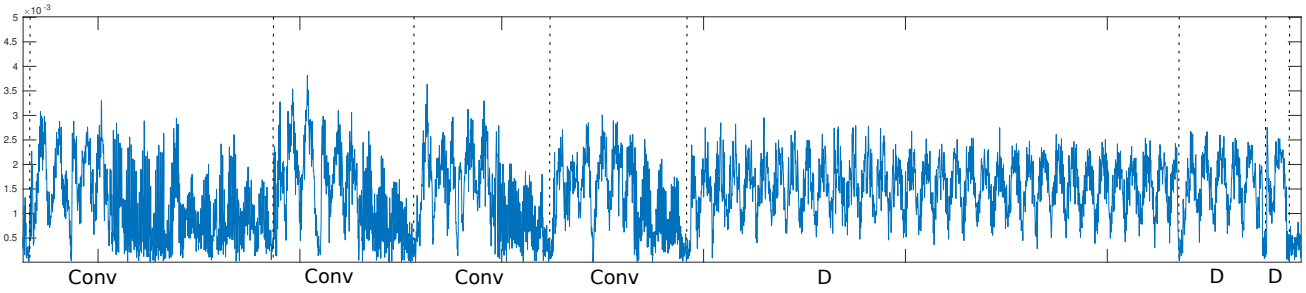
We can reconstruct the topology of a chained DNN by identifying layers from the EM signals of interest. We first analyze the EM traces from the typical CNNs such as VGG and ResNet with layers ranging from convolutional layers, dense layers, batch normalization layers, maxpooling layers, and add layers. The input image resolutions are chosen from $224*224$ and $96*96$. Actually, the larger resolutions the input image has, the stronger and clearer features in EM traces it can induce.

We first identify the skeleton of a VGG16 network from the EM trace with our eyes based on the typical features of different types of layers. Figure 7 is a VGG16 EM trace measured from 2 meter away. We use layer features to recognize the different types of layers. To clearly show the details of EM trace, the entire trace is split into two parts. Figure 7(a) is the first half part of the trace and Figure 7(b) is the second half part of the trace. Layers such as convonlutional layers and dense layers can be easily identified from the trace because of their unique features. Convonlutional layers usually start with high spikes caused by intense data movements in the GPU memory at the beginning of each kernel. An arc shape envelope and a gap at the end of the trace help us to identify the convonlutional layer. The trace of a dense layer is evenly and symmetrically distributed which is different from a trace of convolutional layer. The sharp dips caused by synchronization help us to locate the boundaries between layers. However, layers taking short execution time to compute are hard to identify from the traces. These layers include pooling layer, ReLU layer, softmax layer, and flatten layer, because their GPU memory activities are not as intense as other layers.

The input size also has a significant impact on the shape especially on the length of the trace segments corresponding to a layer. For example, comparing the lengths of the second convoluntional layer with that of the third convoluntional layer in Figure 7(a), the length of the third convoluntional layer is about half of the length of the second convoluntional layer. We can deduce that there is a dimension reduction (such as pooling) after the second convoluntional layer. Layers at the

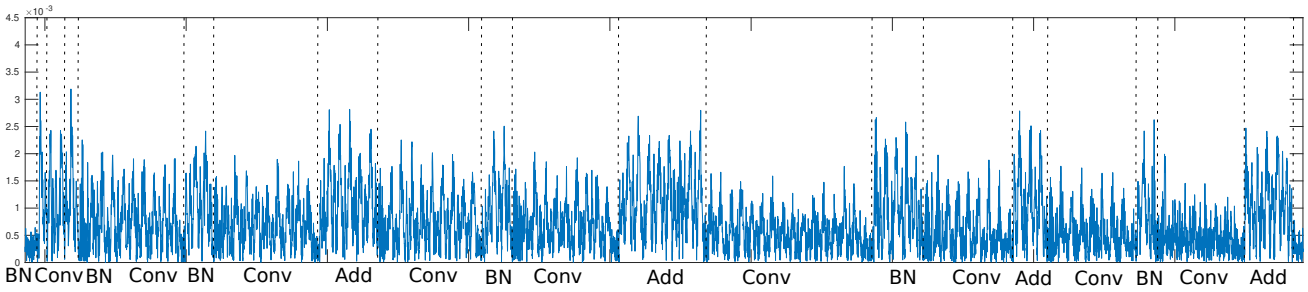


(a) The first half of VGG16 trace

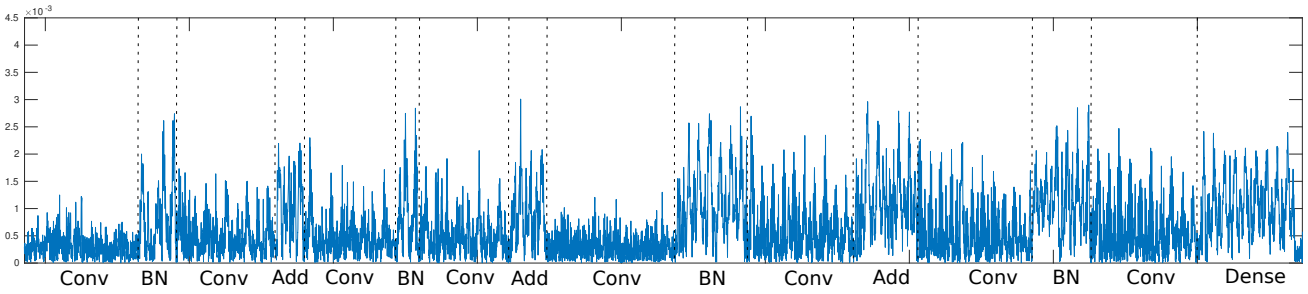


(b) The second half of VGG16 trace

Fig. 7. A VGG16 EM trace. Conv represents a convolutional layer, D represents a dense layer.



(a) The first half of ResNet18 trace



(b) The second half of ResNet18 trace

Fig. 8. A ResNet18 EM trace. Conv represents a convolutional layer, Dense represents a dense layer.

beginning and end of a DNN model are hard to recognize for lack of information from the trace. The first convonlutional layer and the last dense layer only leave short spikes which are caused by short execution time due to the relatively small input tensor sizes. Usually the input size of the first layer of a CNN is small because of a small number of channels (e.g., 3) in the image data. With a narrow sharp spike at the beginning of a trace, we can infer it is a convolutional layer. With a short sharp spike at the end of the trace, we can infer that it is a dense layer with a small input size.

We can also obtain a pooling layer from the lengths of trace segments corresponding to neighboring layers. We can infer

TABLE II
TRACE SEGMENT LENGTH STATISTICS FOR INPUT SIZE $224 \times 224 \times 64$.

kernel size/stride/kernel#	3/1/32	3/2/32	5/1/32	5/2/32	3/1/64
length	14192±363	9658±271	64895±1108	25517±190	27479±276
kernel size/stride/kernel#	3/2/64	5/1/64	5/2/64	3/1/128	3/2/128
length	10047±143	101470±666	26150±300	52760±247	17853±277
kernel size/stride/kernel#	5/1/128	5/2/128	3/1/256	3/2/256	5/1/256
length	197335±4470	47328±147	103770±496	35178±451	279185±1094
kernel size/stride/kernel#	5/2/256	3/1/512	3/2/512	5/1/512	5/2/512
length	92130±128	164091±1513	67564±171	578265±8133	181575±368

TABLE III
TRACE SEGMENT LENGTH STATISTICS FOR INPUT SIZE $96 \times 96 \times 64$.

kernel size/stride/kernel#	3/1/32	3/2/32	5/1/32	5/2/32	3/1/64
length	3564±213	1470±126	28080±540	6280.2±136	5411.2±170
kernel size/stride/kernel#	3/2/64	5/1/64	5/2/64	3/1/128	3/2/128
length	12316±394	29716±631	17306±255	10697±134	16708±385
kernel size/stride/kernel#	5/1/128	5/2/128	3/1/256	3/2/256	5/1/256
length	23257±827	11602±258	20128±249	18495±423	81771±531
kernel size/stride/kernel#	5/2/256	3/1/512	3/2/512	5/1/512	5/2/512
length	32954±568	39326±373	23815±590	146786±4736	46727±578

its existence by comparing the input and output dimension changes from the length of segments. There is a pooling layer with a stride size that is larger than 1 if there is a dimension reduction which can be recognized directly from their length of trace segments. For example, in Figure 7(a), we can clearly recognize that there are dimension reductions in the second, the forth, the seventh, the tenth convolutional layers by comparing their neighboring layers' trace segments. The length reduction of the following layer indicates there is a dimension reduction. This dimension change is usually caused by a pooling.

Short layers such as ReLU activation function, add layer, and pooling are not obvious from the EM trace and can hardly be distinguished from each other. For the activation function, each convolutional layer or dense layer is usually followed by a ReLU activation function unless it is the last layer of the neural network. If it is a sigmoid activation function, it is recognizable due to a longer length.

Figure 8 is the EM trace corresponding to ResNet18 that is measured from 2 meters away. Similar to the VGG16 trace, we can recognize the convolutional layers and dense layer easily with our eyes by locating the unique features of a convolutional layer or a dense layer. The trace segments corresponding to an add layer or a batch normalization layer are also different from that of a convolutional layer or a dense layer. Therefore, an add layer or a batch normalization layer can be distinguished from a convolutional layer or a dense layer. However, they are hard to distinguish from each other because they have similar looks.

The layers are still recognizable when the EM traces are collected with a wall in-between. Figure 9 is a trace corresponding to VGG16 and Figure 10 is a trace corresponding to ResNet18. They are collected behind a 16 cm thick wall (made of drywall). Comparing with the traces shown in Figure 7 and Figure 8, the amplitudes of traces measured behind a wall are attenuated by the obstacles. Nevertheless, with the proposed signal processing techniques, we can still recover the DNN models as there is no wall in-between.

C. Layer Configuration Estimation

The configuration $C = (k, s, n)$ of a commonly used convolutional layer can be recovered by the aforementioned method introduced in Section VI-B. We collect and analyze the trace segment lengths with respect to different configurations. Table II shows the length statistics of input size $224 \times 224 \times 64$. The length instances are represented by the mean and standard deviation of number of samples. From this table, we can see that layers with different configurations can be distinguished from each other due to the different trace segment length distributions. In addition, Table III given the length instances when the input size is $96 \times 96 \times 64$. If there are two instances that are close, we can further analyze the normal distribution of the two instances as shown in Figure 5. From various statistic cases like the ones shown in Table II and Table III, we can map trace segment lengths to different configurations.

VIII. MITIGATION

As mentioned in [1], the EM signal is strong enough to be used to launch a model extraction attack when the input size and batch size are large enough to keep the GPU running at a high-load level. We observe that, with new and powerful NVIDIA GPUs like RTX 2080, when the input image pixels

sizes that are equal or larger than $96 \times 96 \times 3$ with a batch size 1, the EM emanations of interest are noticeable enough to launch a model extraction attack. However, layer features are not distinguishable when the input sizes are smaller (e.g., when image size is $28 \times 28 \times 3$ and batch size is 3). Therefore, we can significantly reduce the input image pixel size first to avoid strong EM emanations of interest. However, this may have a negative impact on the performance of DNN.

The second mitigation approach is to generate some background noises by running some other applications on the GPU. For example, one can co-allocate a small DNN inference process when launching a DNN model for inference. It will consume some additional GPU resources in this case. However, it can add noise into the EM emanations of our interest, which make it harder to precisely extract the DNN structure and the layer configurations.

Another mitigation approach is to insert many short layers (e.g., insert 10 ReLU layers between the third and forth convolutional layers) deliberately without changing the original function of a DNN model. The many short layers inserted at some location can mislead the attacker to derive an incorrect DNN architecture.

IX. Related Work

Model extraction attacks targeting at DNN models try to steal their topology, parameters, hyper-parameters, or functionality. The first type of model extraction attacks use learning-based methods to derive the topology, parameters, hyper-parameters, or the functionality of a DNN model. Knockoff Nets [20] learns a new DNN model with functionality similar to the victim DNN by querying the black-box victim DNN. With the similar method, the work in [19] learns a DNN model by querying the black-box victim DNN. The work in [29] learns the hyper-parameters of regularization term in the objective function of a machine learning models such as regression, logistic regression, support vector machine (SVM). In spite of the significant contributions made in these work, there are still some limitations. These methods either require access to the victim model or the training dataset. The learning based methods also require large amount of computation power and time.

Another type of model extraction attacks is to leverage side-channel information. Such model extraction attacks can be further classified into two categories. The first kind is to use logical side-channel information to infer the DNN model. Cache Telepathy [31] use cache timing side channel to infers dimension sizes. Rendered Insecure [16] infers the model from a co-located process in a cloud environment by monitoring the performance counters of a GPU. DeepSniffer [12] and Hermes [37] extract complex DNN models by snooping the GPU memory bus and/or PCIe bus. The other class is to infer neural network models by leveraging physical side-channel information, especially the EM one. Recent work using EM side-channel information to extract DNN topology and to estimate parameters can be further classified into two categories. The first class is to extract a neural network model from an embedded device or a FPGA as shown in [32], [2] using near-field EM signal. To clear detect the near-field EM emanations from these devices, the devices may need to be decapsulated. The DNN architectures are usually simple due to the limited resource of these devices. The second category as demonstrated in [1] can recover the topology and estimate configurations such as the number of kernels of a complex DNN model from a GPGPU using near-field EM information of the GPU power supply cable. However, they need to have access to the victim machine to launch the attack because their sensor has to be mounted on the power cable of the GPU power supply.

X. Conclusion & Limitation

Conclusion: In this work, we propose CLAIRVOYANCE to stealthily extract the DNN models at a distance by leveraging the EM side-channel information from a GPU. Our method does not need to have access to the victim machine or have physical contact with the victim machine in any form. Our proposed attack can "see" the number of DNN layers and recognize their types. We can also estimate the layer configurations such as the number of kernels, kernel sizes, and strides by simple statistics analysis.

Limitation: This approach cannot steal the parameters of the DNN model. It becomes very hard for our approach to extract a model when the input sizes and the batch sizes are very small. It is also very hard to distinguish very simple layers such as ReLU, batch normalization, and flatten because they have similar features in the trace. Therefore, we need some other heuristics.

Acknowledgment

This work is supported in part by the National Science Foundation (CNS-2147217 and SaTC-2019536). The authors would like to thank the anonymous reviewers for their comments and suggestions that help us improve the quality of the paper.

References

[1] Can one hear the shape of a neural network?: Snooping the GPU via magnetic side channel. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[2] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, 2019.

[3] Robert Callan, Alenka Zajić, and Milos Prvulovic. Fase: Finding amplitude-modulated side-channel emanations. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 592–603. IEEE, 2015.

[4] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[5] Jack Choquette, Edward Lee, Ronny Krashinsky, Vishnu Balan, and Brucek Khailany. 3.2 the a100 datacenter gpu and ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 48–50. IEEE, 2021.

[6] Miro Enev, Sidhant Gupta, Tadayoshi Kohno, and Shwetak N Patel. Televisions, video privacy, and powerline electromagnetic interference. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 537–550, 2011.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[8] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1095–1108, 2017.

[9] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[10] Keith B Hardin, John T Fessler, and Donald R Bush. Spread spectrum clock generation for the reduction of radiated emissions. In *Proceedings of IEEE Symposium on Electromagnetic Compatibility*, pages 227–231. IEEE, 1994.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020.

[13] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[15] Markus G Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *International Workshop on Privacy Enhancing Technologies*, pages 88–107. Springer, 2004.

[16] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 2139–2153, 2018.

[17] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. Eddie: Em-based detection of deviations in program execution. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 333–346, 2017.

[18] Nvidia. Nvidia ampere ga102 gpu architecture. Technical Note V2.1, NVIDIA Corporation, Boise,ID, 2021. https://images.nvidia.com/aem-dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf.

[19] Seong Joon Oh, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer, 2019.

[20] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019.

[21] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[22] Robert Scholtz. The origins of spread-spectrum communications. *IEEE transactions on Communications*, 30(5):822–854, 1982.

[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[26] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[27] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[28] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. Invisible probe: Timing attacks with pcie congestion side-channel.

[29] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE, 2018.

[30] Craig M Wittenbrink, Emmett Kilgariff, and Arjun Prabhu. Fermi gf100 gpu architecture. *IEEE Micro*, 31(2):50–59, 2011.

[31] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[32] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. Deepem: Deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 209–218. IEEE, 2020.

[33] Zihao Zhan, Zhenkai Zhang, and Xenofon Koutsoukos. Bitjabber: The world's fastest electromagnetic covert channel. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 35–45. IEEE, 2020.

[34] Zihao Zhan, Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xenofon Koutsoukos. Graphics peeping unit: Exploiting EM side-channel information of GPUs to eavesdrop on your neighbors. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022.

[35] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS)*, page 159–172, 2018.

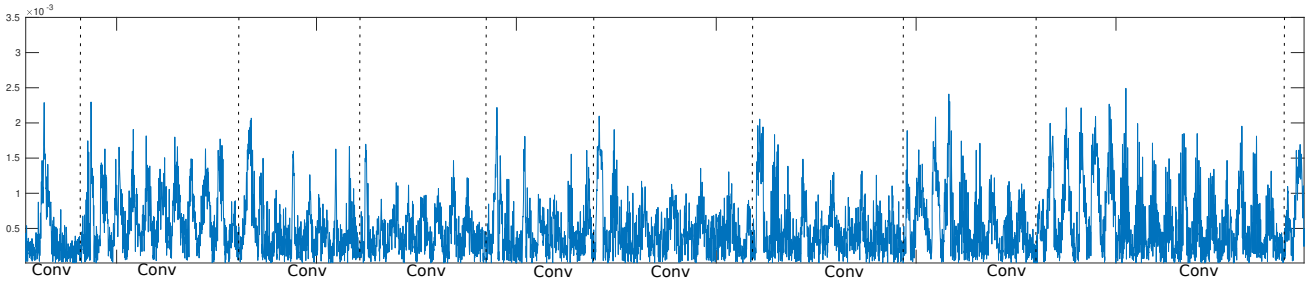
[36] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Bo Li, Peter Volgyesi, and Xenofon Koutsoukos. Leveraging em side-channel information to detect rowhammer attacks. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 729–746. IEEE, 2020.

[37] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. Hermes attack: Steal DNN models with lossless inference accuracy. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

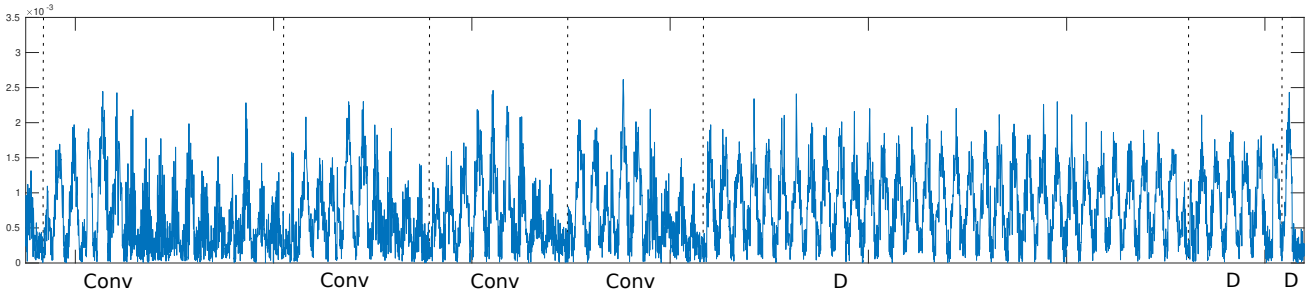
APPENDIX A
ADDITIONAL EXPERIMENTS RESULTS

Figure 9 is a EM trace corresponding to VGG16 collected with a wall in between. The trace is similar to one in Figure 7 when there are no obstacles in between. Therefore, we can directly distinguish the layer types such as convolutional layers and dense layers. We can also deduce the existences of pooling layers from the layer length changes. ReLU layers are not obvious from the trace, but we can guess the existence of a ReLU layer behind each convolutional layer or dense layer (except the last layer) since it is the most often used activation.

Figure 10 is a EM trace corresponding to ResNet18 collected with a wall in between. The trace is also similar to the one in Figure 8 when there are no obstacles in between. Similar conclusions can be drawn from this figure.

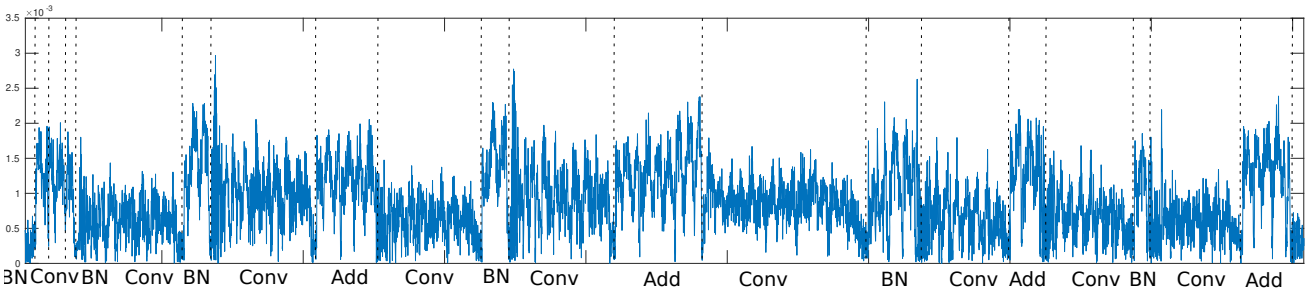


(a) The first half of VGG16 trace over a wall

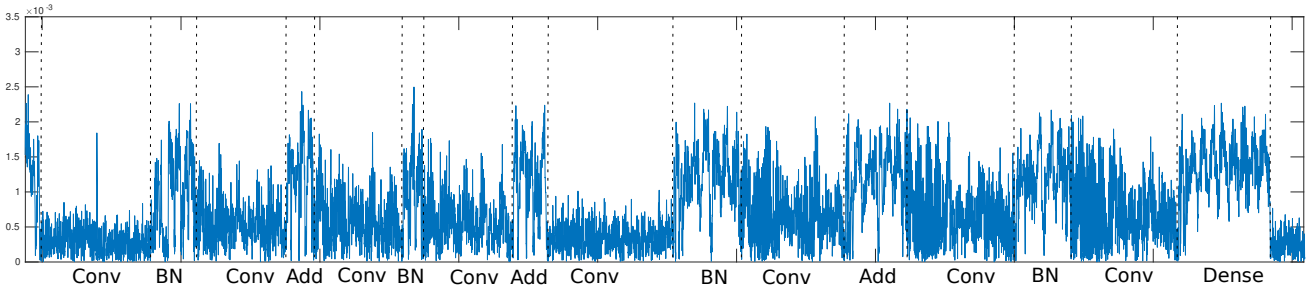


(b) The second half of VGG16 trace over a wall

Fig. 9. A VGG16 EM trace. Conv represents a convolutional layer, D represents a dense layer.



(a) The first half of ResNet18 trace over a wall



(b) The second half of ResNet18 trace over over a wall

Fig. 10. A ResNet18 EM trace over a wall (made of drywall). Conv represents a convolutional layer, Dense represents a dense layer.