IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 1, pp. 78–103. DOI:10.46586/tches.v2025.i1.78-103

TPUXtract: An Exhaustive Hyperparameter Extraction Framework

Ashley Kurian, Anuj Dubey, Ferhat Yaman and Aydin Aysu

Department of Electrical and Computer Engineering, North Carolina State University
akurian,aanujdu,fyaman,aaysu@ncsu.edu

Abstract. Model stealing attacks on AI/ML devices undermine intellectual property rights, compromise the competitive advantage of the original model developers, and potentially expose sensitive data embedded in the model's behavior to unauthorized parties. While previous research works have demonstrated successful side-channel-based model recovery in embedded microcontrollers and FPGA-based accelerators, the exploration of attacks on commercial ML accelerators remains largely unexplored. Moreover, prior side-channel attacks fail when they encounter previously unknown models. This paper demonstrates the first successful model extraction attack on the Google Edge Tensor Processing Unit (TPU), an off-the-shelf ML accelerator. Specifically, we show a hyperparameter stealing attack that can extract *all* layer configurations including the layer type, number of nodes, kernel/filter sizes, number of filters, strides, padding, and activation function. Most notably, our attack is the first *comprehensive* attack that can extract previously unseen models. This is achieved through an online template-building approach instead of a pre-trained ML-based approach used in prior works. Our results on a black-box Google Edge TPU evaluation show that, through obtained electromagnetic traces, our proposed framework can achieve 99.91% accuracy, making it the most accurate one to date. Our findings indicate that attackers can successfully extract various types of models on a black-box commercial TPU with utmost detail and call for countermeasures.

Keywords: Edge TPU · Side Channel · Hyperparameter · Neural Networks · Machine Learning

1 Introduction

Due to the increasing number of connected devices, bandwidth constraints, and the need for high-speed, low-power, and real-time processing of machine learning (ML) models, there has been a significant emphasis on conducting ML inference at the edge [MMH+21]. Dedicated ML accelerators, such as the IBM TrueNorth [ASC+15], Intel Movidius Neural Compute Stick [NCS], Google Edge Tensor Processing Unit (TPU) [TPU], and NVIDIA NVDLA [NVD], have been introduced to meet this demand. Although there are advantages to achieving edge intelligence with such devices, there is also a risk of cyberattacks, and, specifically of, side-channel [CDGK21, NGAD+24] and fault injection attacks [RCYF22].

Given the significant efforts of collecting datasets and training neural networks on ML accelerators or other hardware platforms such as FPGAs, GPUs, and non-commercial custom ASICs, the confidentiality and privacy of the trained models need safeguarding. However, adversaries can still potentially exploit side channel measurements to extract the model details without any knowledge of the internal structures (i.e., black box) of edge ML accelerators [WCJ+21c, GJC23]. These attacks can be categorized into two classes: (i) hyperparameter stealing attacks where the adversary aims to learn the architecture of

Received: 2024-07-15 Accepted: 2024-09-15 Published: 2024-12-09

the trained models such as the types of layers and their configurations, and (ii) parameter stealing attacks where the adversary aims to learn the trained weight and bias values.

A hyperparameter stealing attack followed by parameter extraction can create a high-fidelity substitute model with the extracted information to mimic the victim model [JCB+20]. In a typical neural network, the hyperparameters refer to the model depth, layer type, number of nodes, activation function, strides, padding, number of filters, kernel size, and pool size. The significant reduction in cost and the effort compared to developing a model from scratch motivates an adversary to model stealing through hyperparameters [OSF19]. Model extraction also assists in other attacks like membership inference [SAB+23, SSSS17] and input poisoning [CLL+17].

Despite the existing literature on side-channel-based hyperparameter extraction, all prior works have evaluated a limited search space either with a simple analysis or ML-based modeling [BBJP19, MXL+22, YMY+20, WCJ+21c, DSRB18, YFT20, HZS18, YLX+23, WCJ+21a, HCW+24, GJC23, GLXF23]. As such, *all* hyperparameter stealing attacks focusing on supervised ML-based training fail when encountered with a previously unseen model hyperparameter configuration not included in their training set. Moreover, prior works have largely focused on microcontroller-based designs or FPGA-based solutions where the adversary knows either the software or hardware stack, if not both.

This paper, for the first time, proposes a framework that can perform comprehensive hyperparameter extraction. To that end, we first identify the scalability limitation of existing ML-based approaches for comprehensive extraction that aims to steal previously unseen models. We then propose a new framework, based on *online template matching*, to address the discovered limitation. Finally, we show the application of our framework using electromagnetic (EM) side-channels on a Google Edge TPU—as a corollary of this application, we claim the first hyperparameter stealing attack on Google devices.

We propose a sliding window-based correlation technique performing a layer-wise hyperparameter recovery. Unlike prior works, our attack is capable of extracting all hyperparameters associated with the CNN and MLP layers, such as the `convolution (Conv)`, `fully connected (FC)`, `depthwise convolution (DepthConv)`, and `pooling (Pool)`. Our key observation is that all "offline" modeling-based approaches need to consider the previous layers' hyperparameters to estimate the current layer. This creates a scalability problem as the search space increases. By contrast, performing "online" modeling enables divide-and-conquer one layer at a time at the expense of run-time computations.

Our comprehensive framework discloses a major gap in prior works. These works have exclusively analyzed sequential models comprising a stack of layers, where each layer has exactly one input tensor and one output tensor. However, there are non-sequential/functional models with non-linear topology, shared layers, and even multiple inputs or outputs [HZRS16][non]. Attacking such models is crucial due to their prevalent use in deep learning models. We demonstrate the first attack on non-sequential models. We highlight the challenges of extracting non-sequential models and propose novel techniques to address them.

The main contributions of this paper include the following:

- We develop a framework that comprehensively recovers hyperparameters in run-time with high accuracy. Our attack framework recovers all hyperparameters of `Conv`, `FC`, `DepthConv`, and `Pool` layers that are used in CNN and MLP. This is the first comprehensive framework that offers online templates instead of offline modeling. Our approach addresses the scalability limitation of ML-based modeling.
- We demonstrate the first hyperparameter attack on Google Edge TPUs. We leveraged EM side-channels and exposed unspecified Edge TPU details such as operating frequency and the active locations on the TPU's chip surface as part of our attack.
- We test our hyperparameter extraction framework on deep real-world Edge TPU

models such as MobileNet V3, Inception V3, and ResNet-50. The results show a 99.91% accuracy, which is superior to all prior attacks.

- For the first time, we demonstrate hyperparameter extraction of low-level layers in a non-sequential model such as the `add` and the `concatenate`. We identify the related challenges and requirements for the successful extraction of such layers.

Our research demonstrates that an adversary can effectively reverse engineer the hyperparameters of a neural network by observing its EM emanations during inference, even in a black box setting. The coverage and accuracy of our approach raise significant concerns about the vulnerability of commercial accelerators like the Edge TPU to model stealing in various real-world scenarios.

The rest of this paper is organized as follows. Section 2 discusses the background and threat model of our work. Section 3 exposes unspecified Edge TPU details. Section 4 analyzes the hyperparameter search space and discusses the influence of previous layers and the limitation of using ML for model extraction. Section 5 describes the proposed hyperparameter extraction approach. Section 6 discusses the hyperparameter extraction on non-sequential models. Section 7 evaluates the proposed hyperparameter framework on real-world models. Section 8 discusses the attack transferability, limitations, countermeasures, and future work. Section 9 concludes the paper and Section 10 presents ethical disclosure.

2 Background

2.1 Google Edge TPU

This paper analyzes the side-channel vulnerability of the Google Edge TPU, which is an application-specific integrated circuit (ASIC) designed to execute ML inference at the edge and is widely used in various Google products such as Pixel and Coral [Cora] devices. It delivers high-speed ML computations at the edge. The TPU serves as the core for all Coral devices. Entire inferencing on the Edge TPU is executed using the TensorFlow (TF) Lite libraries. Precompiled TF Lite models are mapped to the hardware using publicly available Edge TPU compilers. The Edge TPU compiler performs various proprietary optimizations to improve performance and energy efficiency.

The Edge TPU was claimed to perform 4 trillion operations per second (TOPS), using 0.5W for each TOPS. Coral offers Edge TPU in various form factors tailored to different prototyping and production settings. These variations range from embedded systems deployed in the field to network systems operating on-premise. For instance, Coral's USB accelerator is a plug-and-play device, while the Dev board is a single-board computer with a removable system-on-module (SoM) featuring the Edge TPU. There are other variations of Coral devices like the Dev board mini, Dev board micro, etc., designed for varying uses.

Figure 1 shows the Coral's Dev board we use in all our experiments [cor20]. The Dev board performs fast inference in a small form factor. For instance, it can execute state-of-the-art mobile vision models such as MobileNet V2 at almost 400 frames per second at low power. The Dev board can be scaled to production by combining the onboard Coral SoM with custom printed circuit board hardware. The SoM provides a fully integrated system, including NXP's iMX 8M system-on-chip (SoC), eMMC memory, LPDDR4 RAM, Wi-Fi, Bluetooth, and Google's Edge TPU coprocessor for ML inferencing. All inferencing with the TPU is executed using TF Lite libraries with Python or C/C++. The Edge TPU's architecture and internal details along with its microarchitecture, instruction set, and compiler remain undisclosed, making it a black box target.

Table 1: Literature summary

Ref.	Attack type	Target	Model information extracted	Limitations
[BBJP19]	EM, timing	ATmega328P, ARM Cortex-M3 microcontroller	MLP: Activation function, number of nodes & layers, weights; CNN: weights	Does not identify `Conv` padding, strides, kernel size, `Pool` size, strides, Relu6 activation; biases are not extracted; parameter extraction on 32-bit MCU is not successful
[ZYC+21]	Power	ZedBoard dual Cortex-A9	Model topology of CNN & MLP; hyperparameters of `FC, Conv` and `Pool`	Conv padding is not analyzed; `Pool` type assumed as MaxPool; unable to identify Relu6, Softmax activation; analysis confined to a few (3) real world models; assumes hyperparameters to be in a limited range
[JMPR23]	EM	ARM Cortex-M7 microcontroller	Model topology of CNN & MLP; hyperparameters of `FC`, `Conv` and `Pool`	Does not evaluate deep models; does not identify the `Conv` activation function, `Pool` type, strides, padding; assumes the pool type as MaxPool; extracts only Relu and Softmax activation functions
[MXL+22]	EM, timing	Nvidia Titan V, Titan X, GTX-1080, GTX-960 GPU	Network topology; activation function	Extracted the hyperparameters of `Conv` layer only; unable to identify Relu6 and Softmax activation
[HLL+20]	EM, bus snooping	NVIDIA K40 GPU	Model topology of DNNs	Hyperparameters are not extracted
[YMY+20]	EM	ZYNQ XC7000 SoC -Pynq-Z1 board	BNN topology; hyperparameters for all layers; weights and biases	Has large search space after model recovery; assumes hyperparameters to be in a limited range
[DCA20]	Power	SAKURA-X FPGA	BNN secret weights	Assumes target architecture and hyperparameters are known
[DSRB18]	Timing	Intel Xeon Gold 5115 server processor	Neural network depth	Limited to known models; does not extract other hyperparameters
[YFT20]	Cache	Dell Precision T1700, 4-core Intel Xeon E3 processor	Model topology of CNN & MLP; hyperparameters of `FC, Conv` & `Pool`	Attack limited to scenarios with shared resources; unable to identify `Conv` strides, `Pool` type, Relu6, Softmax activation
[GFW20]	Timing	Intel i7- 7700 quad-core processor	Weights and biases	Considers parameter recovery only in MLP; does not extract hyperparameters
[HZS18]	Memory access, timing	FPGA accelerator	Model topology of CNN & MLP; zeros in weights of `Conv` layer; hyperparameters of `Conv` & `FC`	Unable to identify #nodes, activation fn, `Pool` layer, and its hyperparameters, `Conv` padding, and stride; attack not scalable to targets that cannot obtain memory access pattern
[GLXF23]	EM	AMD-Xilinx Deep Learning Processing Unit (DPU) Xilinx Zynq ZU3EG	Model topology; weights of first two layers of a CNN	Does not extract hyperparameters of `FC`, `Pool` and `Conv` layer like padding, strides and activation function
[HCW+24]	EM, timing	NVIDIA Jetson Nano	Model topology	Attack cannot be scaled to an unknown model; assumes that the victim model is known
[CW21]	EM, timing	NVIDIA Jetson Nano	Model topology of MLP; hyperparameter of `FC`	Does not target CNNs; attack is performed only on shallow MLP models
[YLX+23]	Power	NVDLA	Model topology	Does not target extraction of hyperparameters of layer types; needs extensive training to generalize the attack
[GJC23]	Power, timing	NVDLA	Model topology of CNN & MLP	`FC` layers are not attacked; `Pool` type, activation functions are not recovered; attack confined to shallow networks; needs training of large number of models
[WCJ+21a]	Cold boot attack	NCS2	Model topology; weights	Does not recover layer type and corresponding hyperparameters; has large search space for model architectures after attack
[WCJ+21b]	EM	NCS2	BNN secret weights	Only a few BNN secret weights are recovered; assumes hyperparameters are known
[WCJ+21c]	Timing	NCS2	Topology of commonly used ML models	Attack confined to known models and hence not scalable
This paper	EM	Google Edge TPU	Topology of sequential & non-sequential MLP, CNNs; hyperparameter of `Conv, FC, DepthConv` and `Pool` layers	Uses more run-time computation

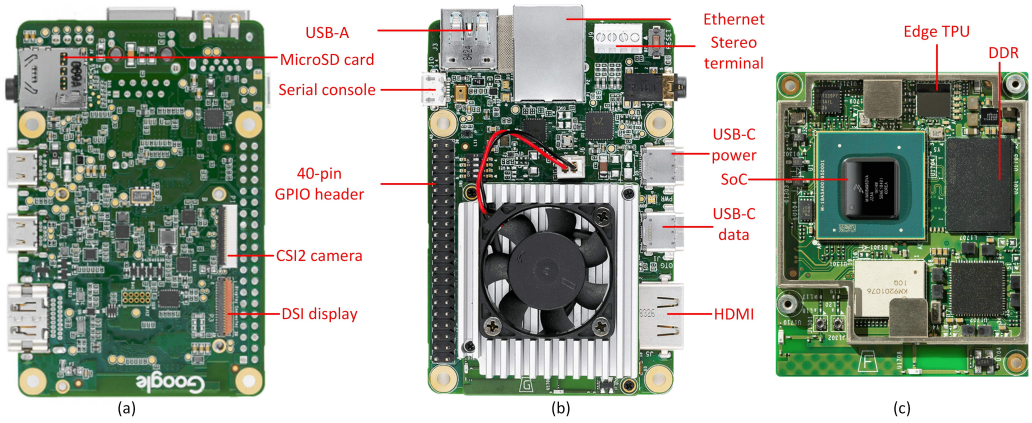

Figure 1: TPUXtract recovers the model hyperparameters from the Edge TPU in the Coral Dev board. The figure shows the back (a), front (b), and SoM (c) of the Coral Dev board. Source: https://coral.ai/products/dev-board/

2.2 Related Work

Side-channel-based reverse engineering attacks on deep neural networks have gained attention due to their capability to extract information about the models, including weights, architecture, hyperparameters, and input data. Prior works show attacks on both general-purpose hardware including FPGAs, microcontrollers, GPUs, and commercial deep neural network (DNN) accelerators—Intel Neural Compute Stick 2 (NCS2) [NCS], and NVDLA [NVD] using various hardware side-channel information such as physical side-channels (e.g., EM, power, timing), off-chip memory access, and resource sharing (e.g., cache and context switching).

We claim that our work is the first to perform an comprehensive hyperparameter extraction. Moreover, this is the first model stealing attack targeting the Google Edge TPU, an ML accelerator. Table 1 shows a list of state-of-the-art attacks categorized based on the type of side-channel leveraged, the target platform, model information extracted, and limitations compared to our work. The table reflects that the majority of the attacks are on microcontrollers [BBJP19, JMPR23], FPGAs [YMY+20, DCA20, ZYC+21, HZS18, GLXF23] and processors [DSRB18, YFT20, GFW20] which are non-commercial ML accelerators. Attacks targeting GPU are also reported but are limited to extracting topology [HLL+20], the hyperparameters of the `Conv` layer only [MXL+22]. Won et al., attacked the NCS2, a commercial ML accelerator but assumed that hyperparameters are known [WCJ+21b]. Other attacks on ML accelerators [GJC23, CW21, HCW+24] recover a few model hyperparameters, *at best a few hundred cases.*

Compared to the previous works, this work is more challenging due to the lack of access to target architecture, operating frequency, instruction set architecture, assembly code, and compiler details. Although there are prior works to extract parameters [BBJP19, WCJ+21b, YMY+20, DCA20] and input [WLL+18] to the DNNs, this work focuses on hyperparameter extraction. Our goal is to comprehensively recover all possible hyperparameters that can feasibly run on the target device. This is the first work that can reverse engineer all the hyperparameters of `Conv, FC, DepthConv`, and `Pool` layers from a hardware accelerator. Also, for the first time, we attack non-sequential models and investigate the extraction of the `add` and the `concatenate` layers.

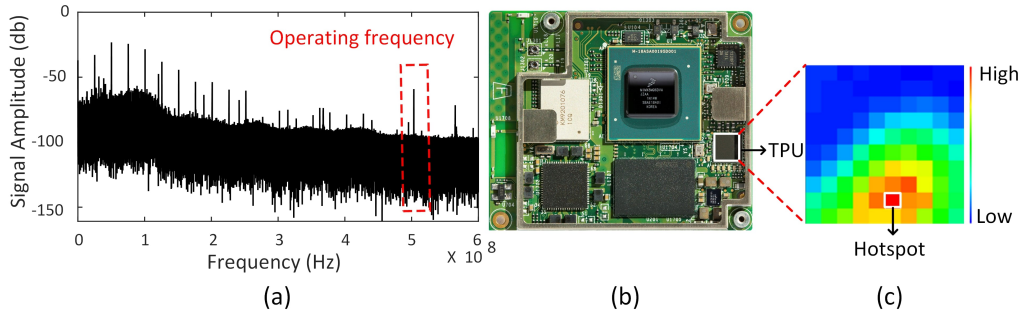

Figure 2: Operating frequency extraction and hotspot identification. The frequency spectrum (a) over the Edge TPU reveals the operating frequency of the target as 500MHz. The hotspot (red tile in (c)) on the TPU is identified by obtaining the heat map (c) after a scan of the TPU mounted on the Dev board SoM (b). The EM probe is then placed over the hotspot for side-channel analysis.

2.3 Threat Model

We follow the typical side-channel analysis assumptions [BBJP19, YMY+20, MXL+22] in hyperparameter extraction—these assumptions also do follow well-known online template attack assumptions where an attacker can swap an unknown model and a known model on the same setup [BCP+19]. The adversary can physically access the target device during inference and capture EM measurements while neural network inferences are underway. We assume adversarial knowledge of the target software deployment environment (TF Lite[1] for Edge TPU), adversarial capability to deploy multiple models on the device, nullify weights/biases of the target model and obtain EM measurements during inferencing [BBJP19]. The attacker aims to exploit the EM side channel for the recovery of model hyperparameters. The adversary is capable of feeding random inputs with known dimensions to the neural network similar to the chosen-plain text attack. However, the adversary does not need to obtain the inference output to reverse engineer the model which makes this work stronger than theoretical model extraction attacks [JSMA19, LM05, WG18, TZJ+16]. Moreover, this work operates in a black-box setting, where the micro-architecture, compiler, and instructions supported by TPU are unknown. The attacker does not possess prior knowledge of model architecture, training dataset, or ML algorithm, making it distinct from theoretical model extraction attacks.

3 Exposing Device Details and Resolving Misalignment

Given the confidentiality of the Edge TPU, the internals and specifications including the operating frequency are undisclosed to the public. In order to conduct hyperparameter extraction, the adversary should first identify the correct IC in the SoM, figure out the operating frequency, and identify the hotspot in the chip that can leak information. The adversary also needs to find reliable patterns on EM traces to obtain well-aligned triggers.

First, we mechanically removed the cooling fan and the heat sink over the SoM to improve the signal quality and for closer placement of the probe over the TPU. Second, we identify the location of the TPU on the SoM using the datasheet [som]. Accurately determining the hotspot on the TPU is imperative to capture EM signals. We ran inference on a CNN (`Conv-MaxPool-FC-FC`), during hotspot identification. We scanned the TPU by dividing the chip surface into a 10 by 10 grid using a motorized XYZ table and collected

[1] Our proposed hyperparameter extraction framework is *not* fundamentally limited to TF Lite framework.

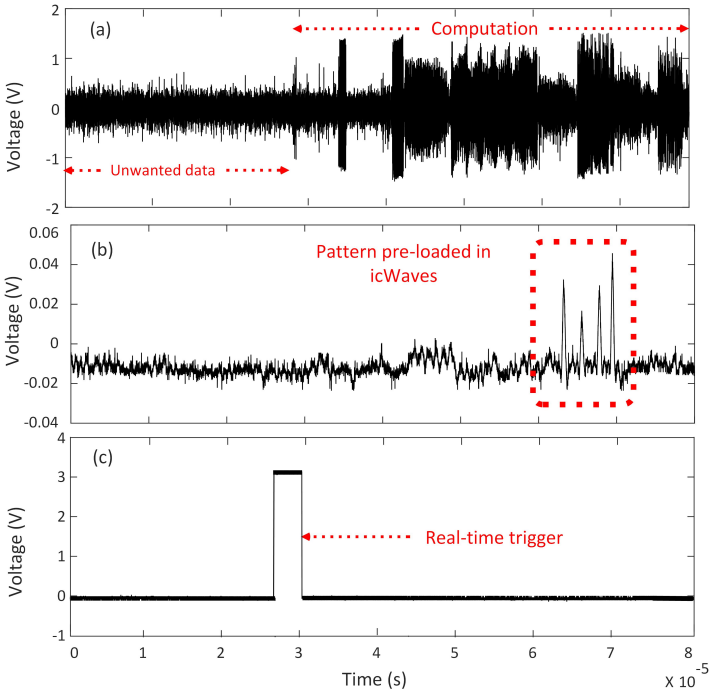

Figure 3: Real-time triggering with icWaves-transceiver setup that surmounts the challenge with triggering and trigger alignment. The raw trace (a) obtained in the absence of real-time triggering captures unnecessary data along with the computations of interest. The raw trace is loaded to the transceiver for frequency modulation to detect unique triggering patterns. A distinct pattern is loaded to the icWaves from the transceiver output (b). The icWaves generates a real-time trigger (c) by matching the pre-loaded pattern with the traces and feeds it to the oscilloscope thereby capturing the desired computations only.

EM measurements in each grid with a high-sensitivity EM probe. Using the probe with the XYZ table makes it possible to automate a scan of the complete chip surface to find the best measurement location. The small step size of the XYZ table provides a very accurate way of measuring the signal intensity on different areas of the target device.

Figure 2 shows the spectral intensity plot over the TPU surface. In our analysis, we disregard lower frequencies due to their allocation for various other functions, as outlined in the datasheet [som]. Specifically, frequencies ranging from 0 to 66MHz are allocated for purposes such as pulse width modulation, 240MHz for WiFi, and 27MHz for HDMI, among others. Consequently, we consider frequencies exceeding 240MHz. The pronounced signal amplitude observed at 500MHz suggests that the operating frequency of the Edge TPU is 500MHz. Under the assumption that Edge TPU has a 64x64 multiply crossbar [BGA+21], our predicted frequency of 500MHz matches the claimed 4TOPS performance. Subsequently, we obtain a heat map of the signals filtered at 500MHz with a 3dB cut-off using a bandpass filter with a frequency range between 498MHz and 502MHz. The heat map gives the signal intensity with increasing intensity from blue to red. The red tile in the heat map is identified as the hotspot and the EM probe is positioned at that location.

We capture EM side-channel traces during inference for hyperparameter recovery. To capture the measurements when the inference starts, we configure a GPIO pin in the TPU as a trigger pin. However, the side-channel analysis may encounter challenges such as excessive data, slow acquisition, and misaligned traces due to large measurement

windows. Addressing these issues involves detecting patterns in the signal prior to the start of measurement. Utilizing a real-time pattern detector and a frequency modulator enables real-time triggering during the acquisition of side-channel measurements. Riscure's icWaves equipment aids in real-time trace alignment by matching signals with pre-loaded patterns by computing the sum of absolute differences. We integrate the icWaves with the transceiver in our setup. The transceiver, equipped with bandpass filters and AM/FM demodulation, overcomes the challenge of detecting unique patterns for triggering in high-frequency switching.

Figure 3*a* displays the raw trace obtained in the absence of triggering. Without real-time triggering, both the computation of interest and unnecessary data are captured, resulting in slow acquisition and misalignment. To address this challenge with triggering, we first pass the raw trace to the transceiver. Figure 3*b* illustrates the frequency-modulated output from the transceiver. Subsequently, a distinct pattern is selected from the transceiver output and loaded into the icWaves, highlighted in red in Figure 3*b*. The icWaves then utilizes this reference pattern to match with the traces in real time. By triggering the oscilloscope at the onset of computations, depicted in Figure 3*c*, the icWaves facilitates the real-time alignment of the computation. Although the pattern match occurs later in the computation, we introduce a -30μs delay to align the trigger with the computation's initiation. Offsetting the trigger timing ensures that the trigger coincides with the beginning of the computation. Further details of the experimental setup are elaborated in Section 7.

4 Analyzing Search Space, Previous-Layer Influence, and Machine Learning Limitations

4.1 Hyperparameter Search Space Analysis

Table 2 enumerates the potential hyperparameters for each layer, selected based on their usage in CNNs on the Edge TPU. The table provides an insight into the total number of possible configurations for each layer. To reverse engineer a single layer effectively, the attacker must consider the sum of the number of configurations for `Conv` (432), `DepthConv` (48), `Pool`(48), and `FC` (5000) layers which result in a total of 5528 possibilities. Although this table shows the sequential models, our actual framework also considers non-sequential ones that further increase the search space. This increase in the search space is a result of considering the `add` and `concatenate` layers. The total search space of 5000+ hyperparameters *per layer* is an order of magnitude more than the most comprehensive prior side-channel based attacks [YMY+20, ZYC+21, BBJP19, MXL+22].

Compared to prior works where the search space is limited to a few hundred per layer, this work comprehensively recovers the hyperparameter by considering all the hyperparameter configurations listed in Table 2. Our approach, despite the large search space per layer, offers a systematic methodology to navigate and extract the hyperparameters efficiently. While these configurations are prevalent, our proposed hyperparameter extraction approach is not confined to these cases and can be extended to accommodate any configuration, e.g., when the proposed framework is extended on a new device with different capabilities.

4.2 Influence of Previous Layers in Hyperparameter Extraction

Since each layer in a model operates on the output of the previous one, the EM signature of a layer is not solely determined by its hyperparameters but also by those of preceding layers. For instance, the input features for a `Conv` layer are the output features of the preceding layer. This makes the attack search space of m^{th} layer in a model n^m, where n is the number of possible hyperparameter configurations for a layer. Therefore, it is not feasible to isolate the hyperparameters of a layer for analysis without accounting for the influence

Table 2: Layer-wise search space analysis for sequential models

Layer type	Hyperparameter	Possible cases	Number of configurations	Total number of configurations (layer-wise)
Conv	Number of filters (F)	8 to 2048 (in power of 2)	9	432
	Kernel size (KS)	2 to 7	6	
	Activation function (Act)	relu, relu6	2	
	Strides (S)	1,2	2	
	Padding (Padd)	same, valid	2	
Pool	Strides	1,2	2	48
	Padding	same, valid	2	
	Pool size (PS)	2 to 7	6	
	Pool type	max, avg	2	
FC	Number of nodes	1 to 1000	1000	5000
	Activation function	relu, relu6, tanh, sigmoid, softmax	5	
DepthConv	Kernel size	2 to 7	6	48
	Activation function	relu, relu6	2	
	Strides	1,2	2	
	Padding	same, valid	2	

of preceding layers. This is not a problem for previous works that do not consider a layer-wise extraction approach [HCW+24]. The works that employ a layer-wise extraction do not *also* face this challenge because they do not aim to recover all hyperparameter configurations resulting in missing scenarios where preceding layers significantly impact the current layer. The accuracy drop in those works can also be a result of the omission of the previous layers' influence.

Given this dependency, the extraction of a layer in a neural network must consider the combined influence of the hyperparameters of both the current and preceding layers. Therefore, it is essential to reverse engineer the hyperparameters of the preceding layers first to extract the hyperparameters of a given layer. This iterative process continues for each subsequent layer, with each layer's hyperparameters being extracted based on the *known hyperparameters of its previous layers.*

To illustrate this concept clearly, consider two neural networks, A and B. Both models consist of two layers in the sequence `Conv-MaxPool`. The input dimension (28x28) and the hyperparameters of the `MaxPool` layer are the same in both models. Figure 4*a* and 4*b* show the side-channel traces corresponding to models A and B, respectively. Initially, extracting the hyperparameters of the first layer can be straightforward since no preceding layers are influencing its pattern. However, upon examining the second layer (`MaxPool`) in both models, we notice differences in their layer signatures despite having identical hyperparameters. This discrepancy arises because the number of filters in the `Conv` layer of model A is 8 and B is 32. As the number of channels in the `Conv` layer (first layer) increases, the number of channels in the input feature to the `MaxPool` layer (second layer) increases. *Therefore, although the second layer of both models has identical hyperparameters, they operate on inputs of different dimensions, leading to large differences in EM trace.* In our example, the input feature to the second layer in model A is of reduced dimension compared to model B. This observation underscores the necessity of considering the influence of previous layers' when extracting the hyperparameters of a given layer. Thus, it is infeasible to analyze a layer's pattern in isolation to extract its hyperparameter as performed in previous ML-based approaches.

Although we show this on a 2-layer network, for a N-layer network the hyperparameter extraction of N^{th} layer does not just rely on hyperparameters of $(N-1)^{\text{th}}$ layer but on earlier layers as well. This is because the input to a layer is not solely determined by its immediate previous layer but by all layers preceding it. For instance, consider two 3-layer

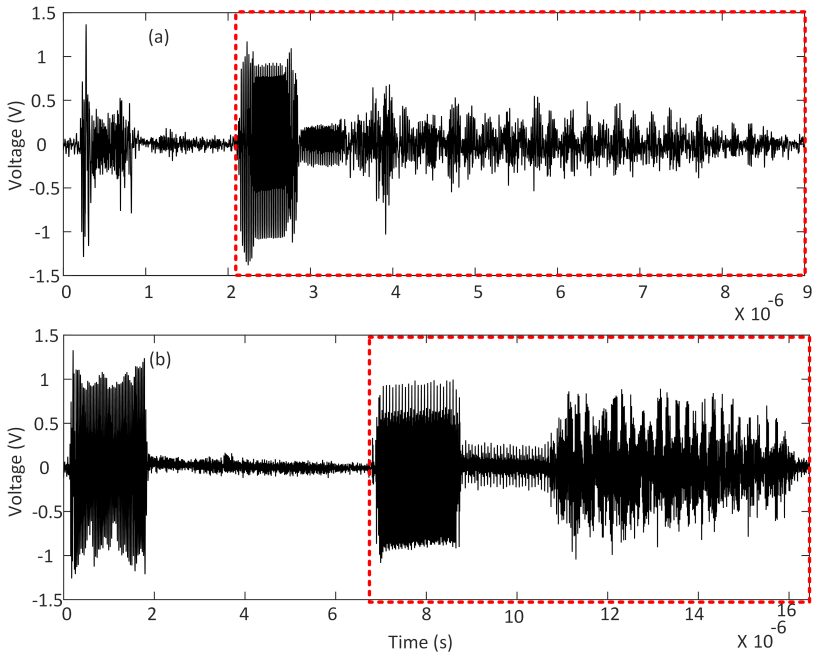

Figure 4: (a) Model A with the layers `Conv-F8-S1-KS2-ActRelu-PaddValid` & `MaxPool-S1-PS10-PaddSame` (b) Model B with the layers `Conv-F32-S1-KS2-ActRelu-PaddValid` & `MaxPool-S1-PS10-PaddSame`. Though the second layer (highlighted in red) in models A and B has identical hyperparameters, they operate on inputs of different dimensions as a result of different hyperparameters of their previous layer. This leads to large differences in their EM trace.

networks with layers in the sequence `Conv-AveragePool-Conv`. The hyperparameters of the second and third layers in both models are identical, while the hyperparameters of the first layer are different. The differing hyperparameters of the first layer will generate input features with different dimensions for the second layer in each network. Consequently, the second layer, acting upon these distinct input dimensions, will produce outputs of different dimensions, impacting the input to the third layer accordingly. Despite the third layer having identical hyperparameters in both networks, it will operate on inputs of different dimensions, leading to variations in their respective side-channel patterns. This example proves that hyperparameters of each layer influence not only the immediate subsequent layer but also propagate their effects through the entire network. Thus, the hyperparameter prediction of N^{th} layer should consider the hyperparameters of all preceding $N-1$ layers.

4.3 Limitation of ML in Hyperparameter Extraction

Prior hyperparameter extraction works have used ML given its high accuracy, though the ML approaches come at the computational cost of training dataset generation and training effort. The dependency of the current layer's activity on previous layers' hyperparameters creates a challenge for pre-trained ML models used in prior works. For example, while the first layer can be modeled with ≈5k classes, the second layer will need ≈25M classes. The computational intensity of using offline ML models for reverse engineering neural network hyperparameters compounds due to the substantial training costs associated with processing a large number of configurations for each layer. Specifically, for a model with K layers, the attacker needs to train for $\prod_{i=1}^{K} \mathrm{S}^i$ classes, where S represents the search

space per layer. Although in practice, this search space can be somewhat mitigated by eliminating less probable configurations, it remains dauntingly large, making it impractical for attackers to brute-force model all potential cases in deep neural networks. Due to this reason, prior works have demonstrated the hyperparameter extraction on a limited number of hyperparameter configurations determined at design time.

Though leveraging offline ML techniques offers promising avenues [YLX+23, HCW+24] for hyperparameter extraction, their practical implementation is hindered by the significant computational demands arising from the comprehensive search space and training complexities. This challenge calls for a different approach to perform comprehensive hyperparameter extraction, which we address with an online template-building approach.

5 Proposed Hyperparameter Extraction Approach

We propose a novel approach for generic hyperparameter extraction based on online template matching with Pearson's correlation distinguisher. Figure 5 illustrates the attack framework. Our key idea is to extract each layer's configuration distinctly, one layer at a time starting from the first layer, by configuring all possible hyperparameters at run-time and by comparing them to the observed victim EM trace with unknown hyperparameters. Once the framework calculates the closest match for the current layer, it will use those hyperparameters to fix the configuration of the current layer and then proceed to search the next layer's hyperparameters until there are no layers left. This enables layer-wise divide-and-conquer and reduces the search space to n, where n is the number of possible hyperparameter configurations for a single layer. This approach ensures that the input to the n^{th} layer matches victim model. Therefore, to extract any layer, the control of inputs to the first layer is sufficient. This is an advantage of our threat model as the adversary can apply random input plaintexts to both victim and template models.

The framework starts by applying random inputs to the TPU with unknown hyperparameters and capturing the EM side-channel trace of the victim model during the inference. The observed model trace initially reflects the input quantization pattern, followed by the layer activity. To isolate the layer pattern, we eliminate the input quantization pattern. In order to find the hyperparameter configuration with the best match, we correlate the generated side-channel traces of all the templates with the victim model trace using Pearson correlation. The resulting correlation plot provides the hyperparameters for the layer and identifies its starting location. The iterative process of template generation for all hyperparameter configurations and the subsequent trace comparison continue until hyperparameters for all layers are extracted. Finally, the predictions for each layer are consolidated to reconstruct the complete model architecture. The components of our proposed extraction approach are discussed in subsequent subsections.

5.1 Eliminate Input Quantization Pattern

The Edge TPU exclusively supports TF Lite models that have undergone full 8-bit quantization and subsequent compilation using the Edge TPU compiler. This quantization process is crucial for enhancing efficiency and reducing model size by converting 32-bit parameter data into 8-bit representation. However, this quantization introduces a distinct pattern in the side channel trace before the layer computations. Given that this input quantization pattern is computationally less intensive compared to layer computations and consistently precedes the first layer, it can be efficiently eliminated using a simple heuristic. Our approach involves discarding samples occurring before the commencement of the first layer. We establish this point using a heuristic where the beginning of the first layer is identified as the first instance where a predetermined threshold is surpassed. Specifically, we set this threshold to be 80% of the maximum signal amplitude observed in the EM

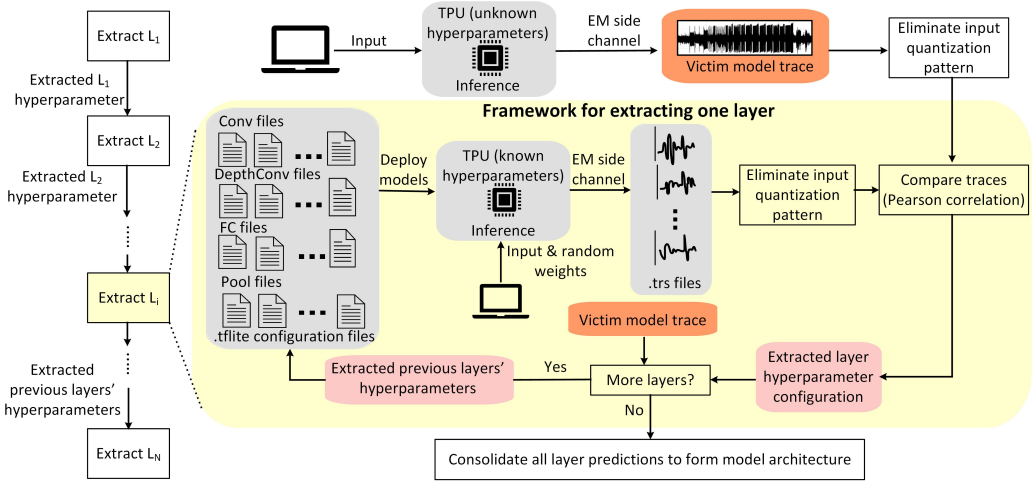

Figure 5: Proposed comprehensive hyperparameter extraction framework. TPUXtract works layer-wise by reconfiguring a device for all hyperparameters to build online templates and by matching those with the observed trace with the unknown hyperparameters successively until all the layers are extracted.

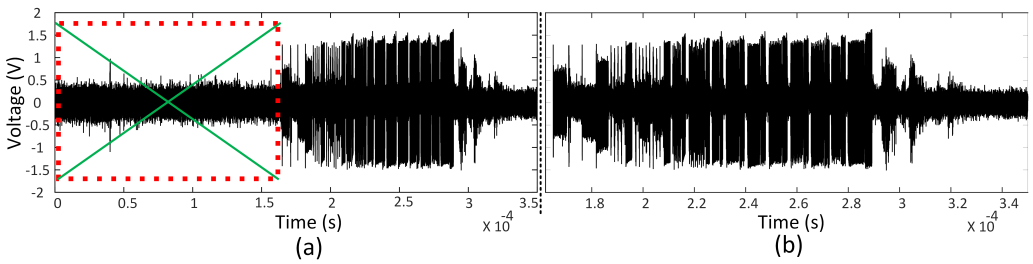

Figure 6: Side channel trace before (a) and after (b) eliminating the input quantization pattern in the MobileNet V1 model. The input quantization starts at the beginning of the trace and ends at the first instance where 80% of the maximum signal amplitude in the EM trace is crossed.

trace. Figure 6*a* and 6*b* depicts the observed side-channel trace of the Mobilenet V1 model before and after applying this heuristic to eliminate the input quantization pattern, showcasing the effectiveness of the method.

5.2 Hyperparameter Prediction using Pearson's Correlation

The similarity between the EM layer pattern template and the observed victim side channel trace is quantified with Pearson's correlation coefficient. To achieve this, layer templates for all possible hyperparameter configurations are correlated with the victim model trace within a chosen window. This window is sufficiently large to encompass the attacked layer. To pinpoint the precise location of the layer beginning, correlation is computed by shifting the templates by one sample within the window. Subsequently, a correlation matrix P is generated using Pearson's correlation, where each row corresponds to a hyperparameter configuration and each column represents the shift within the window. Figure 7 shows the Pearson correlation matrix. An element $P(x, y)$ of the matrix denotes the correlation between the x^{th} hyperparameter template shifted y samples within the window.

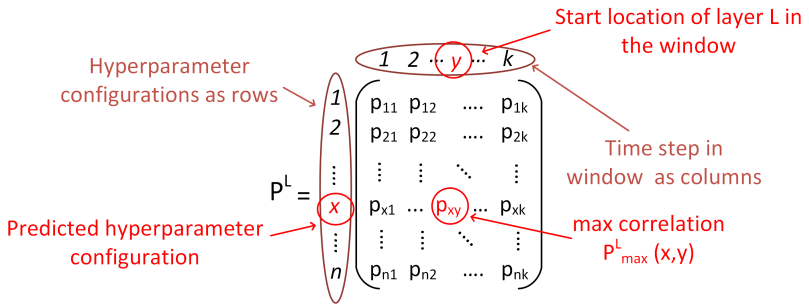

Figure 7: Hyperparameter prediction of a layer L involves correlating templates corresponding to all hyperparameter configurations with the victim trace, thereby generating Pearson correlation matrix P^{L}. The maximum correlation $P^{\mathrm{L}}_{\max}(x, y)$ gives both the hyperparameter prediction and the start location of the layer.

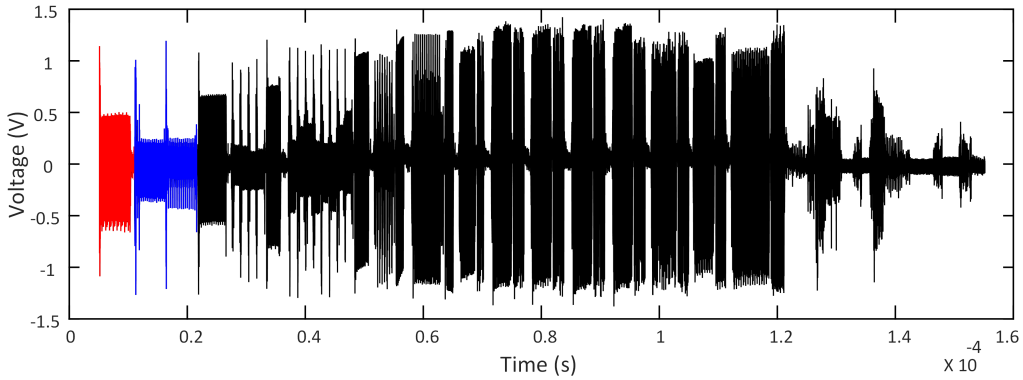

Figure 8: Side channel trace of MobileNet V1 model after eliminating the input quantization pattern; layers 1 and 2 are highlighted in red and blue respectively.

Therefore, hyperparameter prediction for a layer L is the maximum of all elements in the correlation matrix P^{L}, where P^{L} is the matrix obtained by correlating the templates generated for the layer L with the observed trace of the victim model. As a result, the maximum correlation value $P^{\mathrm{L}}_{\max}(x, y)$ not only identifies the hyperparameter of the layer (x) but also determines the location of the layer boundary (y). This process enables precise extraction of layer hyperparameters and their corresponding positions within the trace.

Consider the trace of MobileNet V1 after eliminating the input quantization pattern as in Figure 8. For layer 1, we generate hyperparameter configurations for all the possible cases and acquire their corresponding side-channel trace during inference. Subsequently, we compute the correlation between the observed victim trace and the generated side-channel templates. From the Pearson correlation plot in Figure 9*a*, we identify the first layer as a `Conv` with `filters 8, stride 2, kernel size 3, activation Relu6` and `padding same` (denoted by `Conv-F8-S2-KS3-ActRelu6-PaddSame` in red solid line). The maximum correlation is at step 1, which is the start of the first layer. The beginning of the first layer is marked using a red dotted line in Figure 9*a*. The sinusoidal shape in the Pearson correlation plot is due to the sinusoidal nature of the convolution layer operation. The pattern can be attributed to repeated convolutions on multiple batches of input pixels.

Once we identify the first layer, we generate hyperparameter configurations with the first layer as `Conv-F8-S2-KS3-ActRelu6-PaddSame` and then vary the second layer to have all the possible hyperparameter configurations. The correlation plot in Figure 9*b* correctly

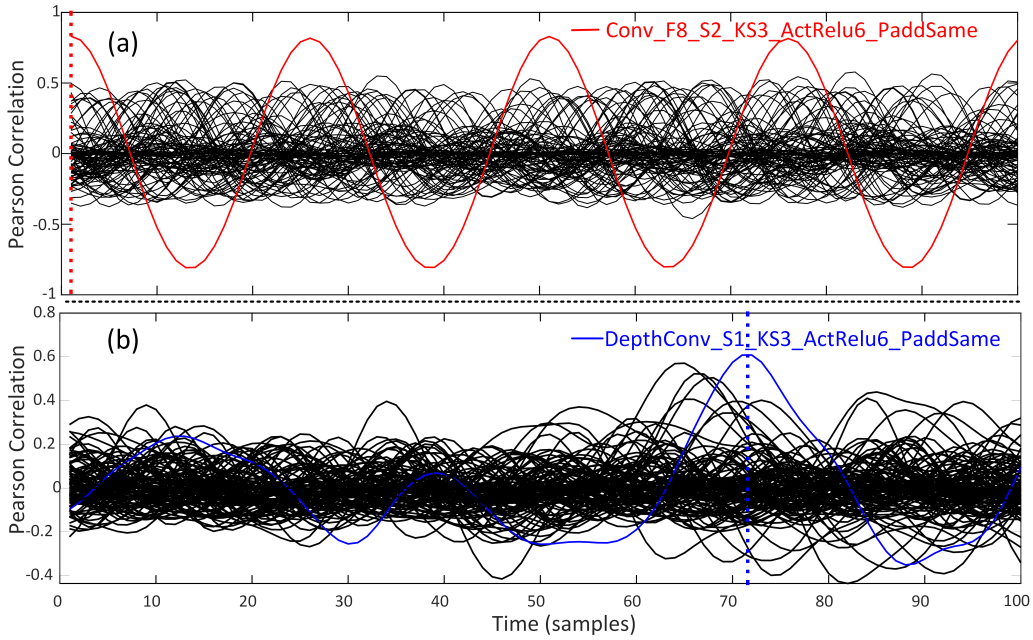

Figure 9: Pearson correlation plot showing the hyperparameter prediction of (a) layer 1 highlighted in red and (b) layer 2 highlighted in blue. The remaining plots in black show the correlation of other hyperparameter configurations. The red and blue dotted line indicates the beginning of the (a) first layer and (b) second layer respectively.

predicts the second layer to be `DepthConv` with `stride 1, kernel size 3, activation Relu6` and `padding same` (denoted by `DepthConv-S1-KS3-ActRelu6-PaddSame` in blue solid line). Additionally, Figure 9*b* shows that the start point of the second layer is at 72 samples from the end of the first layer. The offset between the end of a layer and the beginning of the next layer is accounted for in our framework.

5.3 Identifying the Presence of Additional Layers

The "`more layers?`" block in the framework determines the existence of additional layers based on whether there are remaining samples in the victim model trace after the completion of the previously predicted layer. Therefore, this block takes two inputs: the victim model trace and the endpoint of the previously predicted layer, denoted as Lp. The endpoint of the layer (Lp) is calculated as the sum of the number of samples in the template of Lp and the layer's starting point, which is identified from y in $P^{\text{Lp}}_{\max}(x, y)$. To ascertain the presence of additional layers, we use a heuristic. If the number of samples in the victim trace after the end of Lp exceeds 5000 samples, we predict that more layers exist. The hyperparameter prediction process continues for subsequent layers based on the decision made by the "`more layers?`" block. If the block determines that additional layers exist, the framework generates models while keeping the predictions of the previous layers fixed, followed by the generation of the side channel templates. This iterative approach allows us to systematically explore and analyze the hyperparameter configurations of each layer leading to the extraction of the entire neural network architecture.

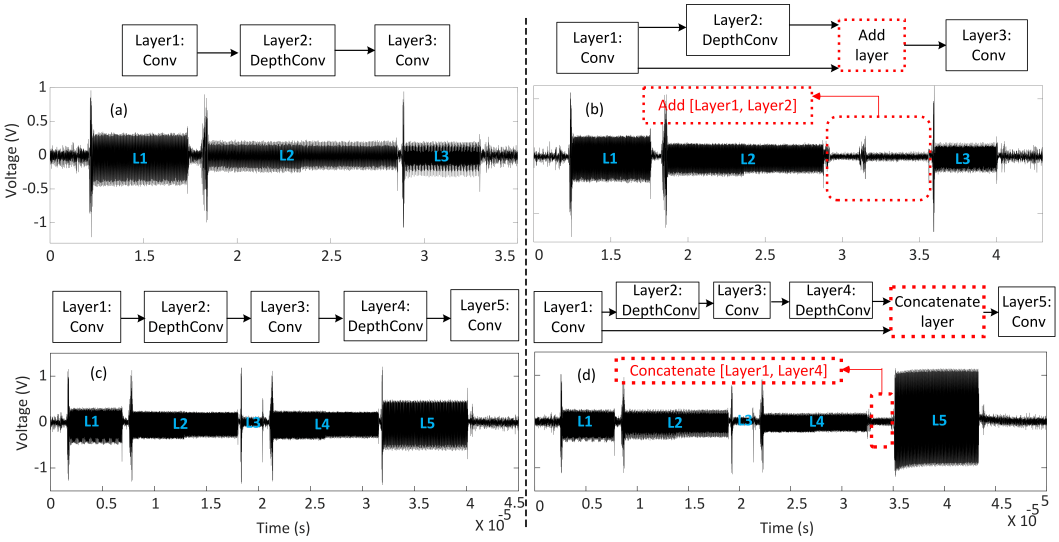

Figure 10: Topology and side channel trace without (a, c) and with (b, d) the `add` and `concatenate` layers, respectively. The introduction of an `add` and `concatenate` layer in a non-sequential model generates a discernible pattern within the side channel trace, thereby facilitating the detection of its presence.

6 Hyperparameter Extraction of Non-Sequential Models

Non-sequential models defy the linear topology of traditional neural networks by incorporating shared layers, multiple inputs, or outputs, resulting in non-linear architectures. While sequential neural networks process layer outputs sequentially, non-sequential models enable more intricate connections where a layer's output can serve as input for multiple subsequent layers. Despite their prevalence in real-world applications, the analysis of hyperparameters for non-sequential models is, unfortunately, omitted in all prior works except one [WZZ+20]. In this study, Wei *et al.* looked at the problem and concluded that shortcuts/connections [HZRS16] in non-sequential layers cannot be directly identified, and argued that attackers may leverage domain knowledge to infer them. Examining the `add` and `concatenate` layers within non-sequential models is pivotal for hyperparameter extraction. These layers operate by combining two input tensors of the same dimensions to produce a unified output tensor. The `add` layer computes element-wise addition, while the `concatenate` layer concatenates inputs along a specified axis.

The first step in extracting non-sequential model hyperparameters is to identify the existence of the `add` and `concatenate` layers (or the lack thereof). Figures 10*a* and 10*b* illustrate the model topology and side-channel trace of neural networks without and with `add` layers, showcasing a distinctive pattern in the trace when an `add` layer is present. Similarly, `concatenate` layers can be identified, as demonstrated in Figure 10*c* and 10*d*. Since the EM signature of the `add` and the `concatenate` layers are fixed, their templates are hard-coded in the hyperparameter framework instead of generating them at run-time. This saves the generation time of their templates for the hyperparameter prediction of each layer. Our heuristic on hard-coded intervals is based on experimental data with varying dimensions. However, the interval does not have to be fixed and could be longer. In such cases, the start of the next layer is still detectable by increased activity at the end of long `add`/`concatenate` layers. This is a minor change to our flow.

However, beyond merely detecting these low-level layers, it is also crucial to ascertain the inputs to them in order to fully characterize the neural network topology. The

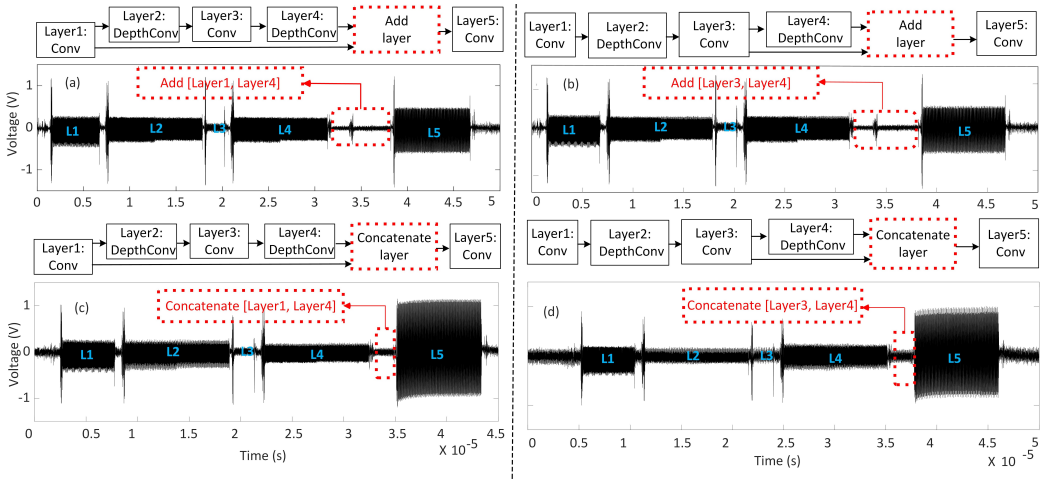

Figure 11: Challenge in identifying the `add` and `concatenate` layer inputs. Though inputs to the `add` layer in (a) and (b) differ—layer 1 and 3 respectively, both give identical side channel patterns making it difficult to infer the `add` layer inputs. Similarly, (c) and (d) show the challenge with `concatenate` layer.

`concatenate` layer in Figure 10*d* receives input from both layer 1 and layer 4 of the model. It is evident that one of the inputs to the `concatenate` layer is from layer 4, as indicated by its pattern immediately preceding the `concatenate` layer in the side channel trace. However, the source of the second input cannot be identified from the side-channel trace, and the input could originate from any layer preceding the `concatenate` layer—specifically, it could be from layer 1, layer 2, or layer 3. Therefore, further investigation is necessary to determine if it is feasible to identify the source of the second input using the available side-channel information.

Figure 11 demonstrates the challenge of identifying the input of the low-level layers in non-sequential models. When one input is fixed to layer 4 and the other input is varied between 1 and 3, there is no clear distinction between the EM traces. Our key insight to resolve this challenge is to nullify the weights of possible input layers in the victim model, one layer at a time, and to observe the reduction (or the lack thereof) of the non-sequential layer's EM traces. The nullification of layers in the victim model can be done by techniques such as controlling the inputs to the first layer [GFW20] and fault injection attacks [BJH+21, TG22]. However, the application of these on the target system is beyond the scope of this work. In subsequent discussion, we explore the efficacy of our technique in both the `add` and `concatenate` layers.

6.1 Inferring `Add` Layer Inputs

We propose a selective deactivation approach to identify the inputs to the `add` layer. The key idea is to zero out the weights one layer at a time and observe traces for a reduction in activity to identify the input layers to the `add` layer. We next formalize our approach in detail and demonstrate it on sample traces. To identify the second input to the `add` layer, we initially hypothesize that the second input originates from layer z, where z represents any potential input layer. Denoting the `add` layer with inputs from layer x and y as `Add`$[L_x, L_y]$, we first identify L_y directly from the EM trace. Next, we nullify the weights and biases associated with both L_y and L_x. Subsequently, we derive the side-channel trace by systematically varying L_x across all possible layer inputs to the `add` layer. These inputs include all layers preceding the `add` layer, excluding L_y.

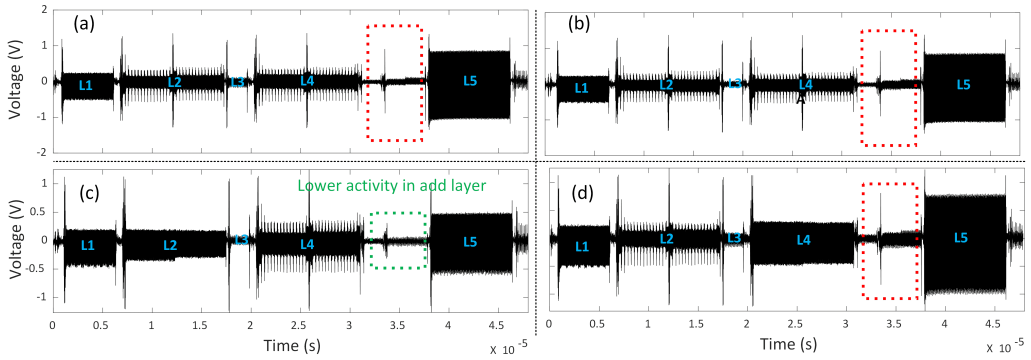

Figure 12: Identifying `add` layer inputs using proposed weight/bias nullifying heuristic. (a) side channel trace for the model when all the layers are activated; (b), (c), and (d) plots correspond to side-channel trace when parameters from layers 2, 1, and 3 are zeroed respectively in addition to layer 4. The reduced amplitude of the `add` layer signature in the plot (c) highlighted in green shows that the second input is from layer 1.

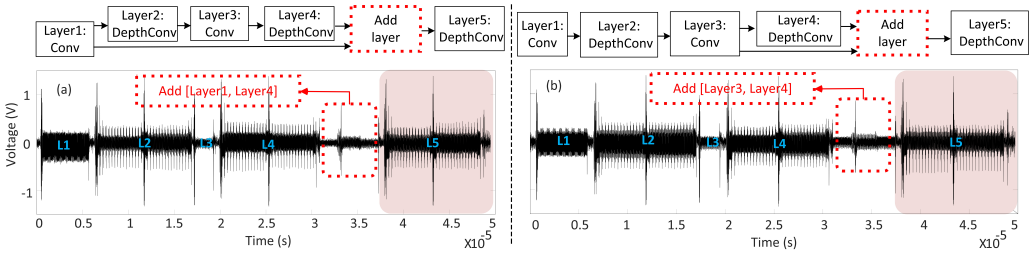

Figure 13: Mispredicting inputs to non-sequential layers does not affect hyperparameter prediction of subsequent layers. Side-channel trace for a model with `Add`$[L_1, L_4]$(a), and `Add`$[L_3, L_4]$(b) gives same pattern for the layer following `add` layer, L_5 (highlighted).

If our hypothesis, `Add`$[L_z, L_y]$, holds true, we anticipate observing diminished activity in the `add` layer signature due to the nullification of both inputs. Conversely, any layer L_x other than L_z would likely result in heightened activity within the `add` layer. If the initial hypothesis proves incorrect, the attacker can select an alternative input layer and repeat the procedure iteratively. This process continues until the chosen layer aligns with the hypothesis, thereby effectively identifying the second input to the `add` layer.

To illustrate this process, consider a neural network with an `add` layer configuration of `Add`$[L_x, L_4]$, where L_x could be layer 1, 2, or 3. To initiate our hypothesis testing, we start with L_1. Figure 11*a* displays the model's topology with `Add`$[L_1, L_4]$. We nullify the weights and biases of L_4 since it is a known input to the `add` layer. Subsequently, we nullify the weights and biases of each potential input candidate for the `add` layer: L_1, L_2, and L_3, and gather their respective side-channel traces. As depicted in Figure 12, the side channel trace obtained for `Add`$[L_1, L_4]$ exhibits lower activity compared to `Add`$[L_2, L_4]$ and `Add`$[L_3, L_4]$, indicating that our hypothesis of L_1 serving as the input to the `add` layer is accurate. This method enables an attacker to discern the model architecture effectively.

6.2 Inferring `Concatenate` Layer Inputs

Similar to the `add` layer, we applied the technique of nullifying weights and biases to deduce the inputs to the `concatenate` layer. However, zeroing the inputs to the `concatenate` layer did not induce any change in the side-channel data. This is attributed to the

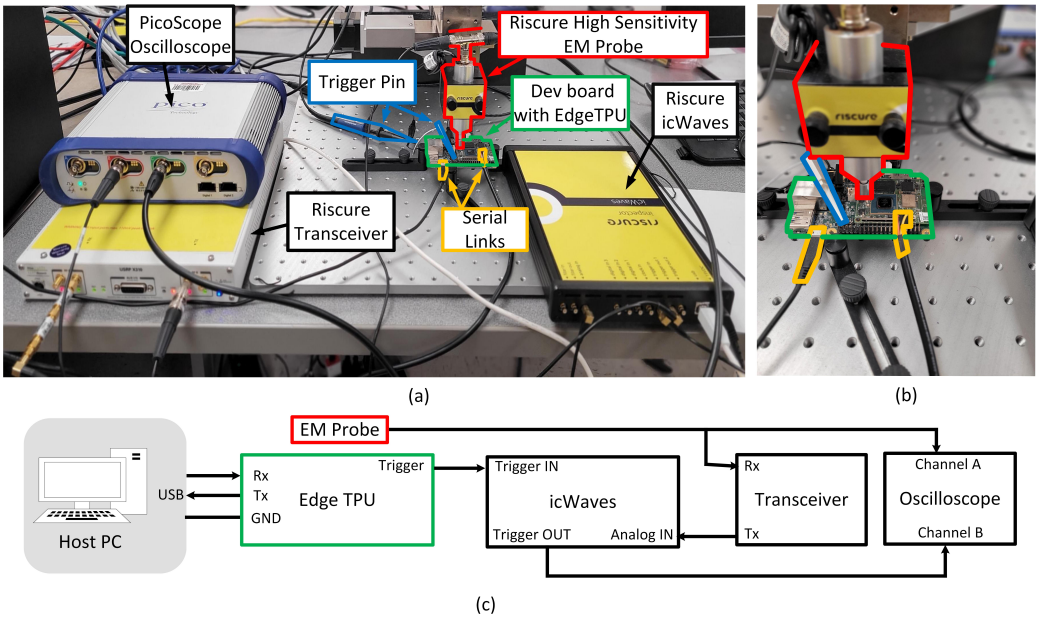

Figure 14: Complete (a) and closer (b) view of the experimental setup. The schematic view (c) shows the signal connections in the setup.

fundamental difference in operations between the `add` and `concatenate` layers. In the `add` layer, the operation entails element-wise addition, resulting in discernible changes in activity within the side-channel trace. Conversely, in the `concatenate` layer, the operation involves simply concatenating the inputs, thereby failing to produce any discernible impact on activity levels within the side-channel trace. Consequently, our method of identifying inputs based on selective deactivation of layers proves ineffective for the `concatenate` layer. However, despite this limitation, an attacker can follow the approach argued in prior work and use domain knowledge to identify the layer shortcuts [WZZ+20]. In our experiments, we assume that the attacker would correctly guess the input connections to `add` and `concatenate` layers. But even when mispredicted, the framework still correctly predicts the next layer's hyperparameters as demonstrated in Figure 13. Despite the different inputs to the `add` layer in Figure 13*a* and 13*b* being L_1 and L_3, respectively, the EM pattern of L_5 (`DepthConv`) remains unchanged. Additionally, Figures 11 and 12 generalize this for a different layer type (`Conv`). These results indicate that the inputs to the non-sequential layers do not influence the hyperparameters of the subsequent layers.

7 Evaluation of TPUXtract on Real-World Models

7.1 Experimental Setup

The target platform used in all our experiments is the Coral Dev board [cor20] featuring the Edge TPU. Figure 14*a* and 14*b* show the complete and closer view of the experimental setup respectively. Riscure's EM probe station which has the the high sensitivity EM probe and a motorized XYZ table is used for acquiring the EM emanations. This setup is integrated with the Inspector software for configuration, taking measurements and subsequently performing analysis. The high-sensitivity probe is capable of measuring weak emanations from small current loops in the chip and the XYZ table is used to automate a scan of the complete chip surface to find the best measurement location. The traces are

captured using a Picoscope 6000E high-end oscilloscope and aligned real-time using the icWaves and the transceiver.[2] The noise in the input signal to the transceiver is attenuated by 30dB using 10dB and 20dB attenuators in series.

In order to improve the signal quality and for closer placement of the probe to the TPU, we removed the cooling fan over the SoM mechanically.[3] The setup requires flashing Mendel Linux to the board and then accessing the board's shell terminal [dev]. This does not require privileged or root access to the device. A GPIO pin of the Edge TPU is configured as the trigger pin. This is a coarse-grained trigger used to signal the beginning of the inference phase (details of triggering are discussed in Section 3). The UART pins of the board are used for serial communication between the TPU and the host PC. [4]

Figure 14*c* shows the schematic diagram of the experimental setup. The Edge TPU takes input from the host PC for inference. The TPU sends the trigger signal to the icWaves to signal the start of the inference. The EM probe sends collected traces to the transceiver. The transceiver after frequency modulation sends the traces to the icWaves for alignment. The icWaves obtains the measurements from the transceiver upon receiving the trigger from the TPU. The icWaves trigger the oscilloscope in real time to capture EM measurements from TPU. The host machine synchronizes operations and collects the measurements from the oscilloscope for hyperparameter extraction.

7.2 Results

We employ the Operation Error Rate (OER) as a metric to assess the effectiveness of the attack [LGL+21, YLX+23]. The OER is computed as $L(s', s)/||s||$, where $||s||$ represents the length of the sequence s, and $L(s', s)$ denotes the edit distance (Levenshtein) between the predicted operation sequence s' and the ground-truth sequence s. A lower OER indicates higher accuracy. In our study, we evaluate the hyperparameter extraction on TF Lite models [5] optimized for Edge TPU[6], as documented in [corb].[7] These models are specifically designed for real-world computer vision applications and are engineered to accommodate multiple input dimensions. Our assessment involves running inference on all models outlined in Table 3. Our attack necessitates only a single EM trace per template during inference. Table 3 presents a detailed breakdown of the models examined, including their respective applications, OER, and corresponding hyperparameter extraction accuracy.

Our attack demonstrates a high success rate, averaging 99.91%. Predominantly, the hyperparameters are accurately predicted, with only minor discrepancies observed in one or two hyperparameter instances within a model across a subset of models. Notably, these discrepancies do not entail drastic mispredictions, such as an entire layer being incorrectly inferred, but rather entail slight deviations. For example, in the MobileNet V2 model, there are 63 layers and a total of 308 hyperparameters. We can predict all 62 layers accurately, but in one of the layers, while the hyperparameter is an `FC` with 965 nodes, we detect it as an `FC` with 963 nodes. Moreover, our experiments show that a misprediction in one layer does not affect hyperparameter predictions in subsequent layers. If necessary, error cascading can be addressed by monitoring the correlation score and adding an error

[2] Typically, a layer has ∼6000 to 10,000 samples at a 1.25GHz/s sampling rate, resulting in $6000 \times N$ to $10000 \times N$ samples for N layers. EM traces are captured as .trs files, exported to .csv, and processed in MATLAB. Our framework is effective with a single trace, improving computation time and storage.

[3] We used an external cooling fan to cool the device. This fan is not included in the experimental setup, Figure 14a because of its size.

[4] We use an AMD Ryzen 9 7950X 16-core processor 4.50GHz host PC for the computations.

[5] Edge TPU does not support batch normalization; hence, it is not included in our templates. TPU might be doing internal normalization but this does not affect our attack accuracy.

[6] Edge TPU does not support batch size greater than one during inference due to dimensional constraints.

[7] Our framework takes into account only the operations supported by TPU. The evaluation is on models pre-trained for inference on TPU [corb]. The operations like `hard-swish` and `Conv2DTranspose` are not supported by Edge TPU. From the ground truth, it seems like a `hard-swish` is implemented with `ReLU6` and `Conv2DTranspose` with `Conv2D`.

check with alternative guesses if the score falls below a certain threshold. In summary, our conducted attack exhibits remarkable efficacy and coverage in reverse-engineering neural networks, making them superior to all prior efforts.

Table 3: Attack accuracy on real-world models offline-trained for Edge TPU

Application	Model	N/S	No. of layers	Layer types	OER	Accuracy
Image classification	EfficientNet-EdgeTpu (S)	N	63	C,F,D,A	0.317	99.68%
	Inception V1	N	80	C,P,Ct	0	100%
	Inception V3	N	95	C,P,D,Ct	0	100%
	MobileNet V1	S	28	C,P,D	0	100
	MobileNet V2	N	63	C,P,F,D,A	0.324	99.67%
	MobileNet V3	N	75	C,D,A	0	100%
	ResNet-50	N	75	C,P,F,A	0.584	99.4%
	Popular Products V1	N	70	C,P,F,D,A,Ct	0	100%
Object detection	SSD MobileNet V2	N	104	C,D,A,Ct	0	100%
	SSDLite MobileDet	N	130	C,A,D,Ct	0	100%
	EfficientDet-Lite0	N	242	C,P,D,A,Ct	0	100%
Semantic segmentation	U-Net MobileNet v2	N	79	C,D,A,Ct	0.294	99.7%
	MobileNet v2 DeepLab v3	N	68	C,P,D,A,Ct	0	100%
	MobileNet v1 BodyPix	N	37	C,D,Ct	0	100%
	ResNet-50 BodyPix	N	83	C,P,A,Ct	0	100%
Pose estimation	PoseNet MobileNet V1	N	35	C,D,Ct	0	100%
	MoveNet.SinglePose.Lightning	N	155	C,D,A,Ct	0	100%
	PoseNet ResNet-50	N	80	C,P,A,Ct	0	100%

N:Non-sequential, S:Sequential, C:Conv, P:Pool, F:FC, D:DepthConv, A:Add, Ct:Concatenation

8 Discussion

8.1 Faster Search Space Exploration

The efficiency of our hyperparameter extraction framework can be enhanced by expediting the exploration of the search space. The hyperparameter search space for each layer, encompassing all potential configurations, amounts to 5530 possibilities. Our online template matching approach necessitates approximately 3 hours ($\approx$2.75 hours for template generation and $\approx$0.25 hours for hyperparameter prediction) to predict the hyperparameters of an individual layer. Consequently, for a model comprising k layers, the process of reverse-engineering the entire set of hyperparameters entails a computational duration of approximately $3 \times k$ hours. This computational overhead can be mitigated by faster exploration of the search space per layer. A possible approach for faster search space exploration could be to develop an algorithm that iterates through likely hyperparameter cases first, thus bypassing unlikely scenarios. The framework may then consider the improbable cases only if the correlation of predicted hyperparameters is low. To identify the likely hyperparameter configurations, several strategies can be employed:

- Utilize common trends observed among most models, such as ensuring that a `Pool` layer cannot be a first layer in a neural network, consecutive `Pool` layers are uncommon, and a `Conv` or `Pool` layer typically succeeds a `Conv` layer.
- Employ dynamic adjustment of the search space by eliminating unlikely cases or prioritizing the most probable scenarios based on preceding layer predictions within a given model. For instance, if a pattern emerges in a model, such as alternating `Conv` and `DepthConv` layers or the presence of shortcuts after every third layer, the algorithm can adapt accordingly.

- Incorporate domain knowledge into the algorithm. For example, if the model architecture is known to be restricted to MLP, only `FC` layers need to be considered.

Therefore, the development of a faster search space exploration algorithm represents a promising avenue to accelerate our hyperparameter extraction framework. By intelligently navigating through likely scenarios and adapting to individual model characteristics, this algorithm can reduce the hyperparameter extraction time while preserving accuracy.

8.2 Attack Transferability

Our framework is versatile for extension to operate across various devices and software deployment environments. The requirements for attack transferability are the adversarial knowledge of the target software deployment environment, attacker's capability to capture side-channel information during inference and generate templates online. By contrast, prior ML-based hyperparameter extraction methodologies necessitate the creation of new training datasets for each unique device, model, or layer type. Our framework incorporates new layers or model types by merely entailing their addition into the existing search space, followed by the generation of corresponding templates. Moreover, the adversary can assess the effectiveness of the heuristics employed in the attack framework for a new target; otherwise, they can refine these heuristics accordingly. Our framework sets certain thresholds, e.g., for eliminating input quantization pattern and identifying the presence of additional layers, which may require dynamic generation at runtime if patterns vary for a new target. Thus, with minor adjustments, attackers can adapt our framework for hyperparameter extraction. This inherent adaptability ensures that our approach remains agnostic to the specific nuances of the device, model, or layer types encountered, rendering it broadly applicable across diverse neural network architectures and hardware platforms.

8.3 Extraction Countermeasures

The following countermeasures can be pursued to enhance the security of neural networks against the proposed hyperparameter extraction attack:

- The attack framework eliminates the input quantization pattern and identifies the beginning of the first layer when 80% of the maximum signal amplitude is crossed for the first time. A simple countermeasure is to add dummy operations during input quantization to increase the activity above the threshold. This can cause desynchronization resulting in incorrect identification of the layers and hyperparameters.

- Adding dummy operations between random layers can introduce activity in the EM side-channel trace at different locations each time depending on when the dummy operations are performed. Another approach is to dynamically alter the layer execution order with each inference run. Both techniques result in desynchronization, causing the template-matching framework for hyperparameter extraction to fail.

- Running random concurrent operations on TPU during inference can introduce noise which makes the layer signatures less discernible making the approach fail.

- Utilizing ensemble methods, such as random forests, can aggregate predictions from multiple individual models [TZJ+16]. This aggregation can introduce additional noise into the side-channel activity, further obfuscating the layer signatures.

- Combining layers on TF Lite files for obfuscation can be an effective countermeasure [ZGW+23]. However, this requires modifying deep learning libraries to maintain correct model computations.

8.4 Limitations

Although we highlight various limitations of our approach throughout the paper, here we provide a summary: (i) The attack is limited to extracting hyperparameters and does not target the extraction of model parameters. (ii) The hyperparameter extraction framework requires approximately 3 hours of run-time computation per layer. (iii) To identify the inputs to non-sequential layers, the attacker needs to nullify the weights and biases of the input layers. However, our heuristic approach for nullifying parameters to identify the inputs to the `concatenate` layer is not feasible.

9 Conclusion

Neural networks represent valuable intellectual properties that are susceptible to model theft attacks, even in black box scenarios. In this work, we present the first comprehensive neural network hyperparameter extraction attack and we target the Google Edge TPU for the first time through EM side-channel analysis. Leveraging an online template-matching approach, we comprehensively recover all hyperparameters of CNN and MLP networks. Our evaluation, conducted on real-world models such as MobileNet V3, Inception V3, and ResNet-50, offline-trained on the Edge TPU, showcases an accuracy of 99.91%. Furthermore, we unveil undisclosed device details such as operating frequency and hotspot location. Moreover, for the first time, we demonstrate the extraction of low-level layers such as `add` and `concatenate` in non-sequential models. Our results show the attacker's capability to reverse-engineer different models from a commercial black-box TPU, highlighting the need for defenses. The proposed framework can be enhanced by faster search-space exploration.

10 Ethical Disclosure

The vulnerability has been disclosed to Google.

Acknowledgements

We thank Olivier Benoit, Elke De Mulder, Aria Shahverdi, Daniel Moghimi, Sanjeev Das, and Mike Tunstall from Google for their feedback and helpful discussions. This project is supported in part by NSF Award No. 1943245 and Google Research Scholar program.

References

[ASC+15] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

[BBJP19] 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.

[BCP+19] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. *Journal of Cryptographic Engineering*, 9:21–36, 2019.

[BGA+21] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 159–172. IEEE, 2021.

[BJH+21] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. SNIFF: Reverse Engineering of Neural Networks With Fault Attacks. *IEEE Transactions on Reliability*, 71(4):1527–1539, 2021.

[CDGK21] Hervé Chabanne, Jean-Luc Danger, Linda Guiga, and Ulrich Kühne. Side channel attacks for architecture extraction of neural networks. *CAAI Transactions on Intelligence Technology*, 6(1):3–16, 2021.

[CLL+17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[Cora] Coral. https://coral.ai/. Accessed: 2024-07-15.

[corb] coral AI models. https://coral.ai/models/. Accessed: 2024-07-15.

[cor20] Google Coral. Dev board. https://coral.ai/products/dev-board/, 2020. Accessed: 2024-07-15.

[CW21] Łukasz Chmielewski and Léo Weissbart. On Reverse Engineering Neural Network Implementation on GPU. In *Applied Cryptography and Network Security Workshops: ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21–24, 2021, Proceedings*, pages 96–113. Springer, 2021.

[DCA20] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. MaskedNet: The First Hardware Inference Engine Aiming Power Side-Channel Protection. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 197–208. IEEE, 2020.

[dev] Getting started with Dev board. https://coral.ai/docs/dev-board/get-started/#requirements. Accessed: 2024-07-15.

[DSRB18] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E Balas. Stealing Neural Networks via Timing Side Channels. *arXiv preprint arXiv:1812.11720*, 2018.

[GFW20] Cheng Gongye, Yunsi Fei, and Thomas Wahl. Reverse-Engineering Deep Neural Networks Using Floating-Point Timing Side-Channels. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[GJC23] Naina Gupta, Arpan Jati, and Anupam Chattopadhyay. AI Attacks AI: Recovering Neural Network architecture from NVDLA using AI-assisted Side Channel Attack. *Cryptology ePrint Archive*, 2023.

[GLXF23] Cheng Gongye, Yukui Luo, Xiaolin Xu, and Yunsi Fei. Side-Channel-Assisted Reverse-Engineering of Encrypted DNN Hardware Accelerator IP and Attack Surface Exploration. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023.

[HCW+24] Péter Horváth, Lukasz Chmielewski, Leo Weissbart, Lejla Batina, and Yuval Yarom. CNN Architecture Extraction on Edge GPU. In *International Conference on Applied Cryptography and Network Security*, pages 158–175. Springer, 2024.

[HLL+20] 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.

[HZRS16] 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.

[HZS18] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[JCB+20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High Accuracy and High Fidelity Extraction of Neural Networks. In *29th USENIX security symposium (USENIX Security 20)*, pages 1345–1362, 2020.

[JMPR23] Raphaël Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud. Like an Open Book? Read Neural Network Architecture with Simple Power Analysis on 32-Bit Microcontrollers. In *International Conference on Smart Card Research and Advanced Applications*, pages 256–276. Springer, 2023.

[JSMA19] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. PRADA: Protecting Against DNN Model Stealing Attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

[LGL+21] Xiaoxuan Lou, Shangwei Guo, Jiwei Li, Yaoxin Wu, and Tianwei Zhang. NASPY: Automated Extraction of Automated Machine Learning Models. In *International Conference on Learning Representations*, 2021.

[LM05] Daniel Lowd and Christopher Meek. Adversarial Learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647, 2005.

[MMH+21] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine Learning at the Network Edge: A Survey. *ACM Computing Surveys (CSUR)*, 54(8):1–37, 2021.

[MXL+22] Henrique Teles Maia, Chang Xiao, Dingzeyu Li, Eitan Grinspun, and Changxi Zheng. Can one hear the shape of a neural network?: Snooping the GPU via Magnetic Side Channel. In *USENIX Security Symposium*, pages 4383–4400, 2022.

[NCS] Intel. 2018. Neural Compute Stick 2. https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html. Accessed: 2024-07-15.

[NGAD+24] Tushar Nayan, Qiming Guo, Mohammed Al Duniawi, Marcus Botacin, Selcuk Uluagac, and Ruimin Sun. SoK: All You Need to Know About On-Device ML Model Extraction-The Gap Between Research and Practice. 2024.

[non] Functional apis for non-sequential models. https://keras.io/guides/functional_api/. Accessed: 2024-07-15.

[NVD] NVIDIA Deep Learning Accelerator. http://nvdla.org/. Accessed: 2024-07-15.

[OSF19] 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.

[RCYF22] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In *2022 IEEE symposium on security and privacy (SP)*, pages 1157–1174. IEEE, 2022.

[SAB+23] Shubhi Shukla, Manaar Alam, Sarani Bhattacharya, Pabitra Mitra, and Debdeep Mukhopadhyay. "Whispering MLaaS": Exploiting Timing Channels to Compromise User Privacy in Deep Neural Networks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 587–613, 2023.

[som] Coral Dev board SoM datasheet. https://coral.ai/static/files/Coral-SoM-datasheet.pdf. Accessed: 2024-07-15.

[SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[TG22] Shahin Tajik and Fatemeh Ganji. Artificial Neural Networks and Fault Injection Attacks. In *Security and Artificial Intelligence: A Crossdisciplinary Approach*, pages 72–84. Springer, 2022.

[TPU] Edge TPU. https://cloud.google.com/edge-tpu. Accessed: 2024-07-15.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[WCJ+21a] Yoo-Seung Won, Soham Chatterjee, Dirmanto Jap, Arindam Basu, and Shivam Bhasin. DeepFreeze: Cold Boot Attacks and High Fidelity Model Recovery on Commercial EdgeML Device. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[WCJ+21b] Yoo-Seung Won, Soham Chatterjee, Dirmanto Jap, Arindam Basu, and Shivam Bhasin. WaC: First Results on Practical Side-Channel Attacks on Commercial Machine Learning Accelerator. In *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, pages 111–114, 2021.

[WCJ+21c] Yoo-Seung Won, Soham Chatterjee, Dirmanto Jap, Shivam Bhasin, and Arindam Basu. Time to Leak: Cross-Device Timing Attack On Edge Deep Learning Accelerator. In *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–4. IEEE, 2021.

[WG18] 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.

[WLL+18] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406, 2018.

[WZZ+20] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. Leaky DNN: Stealing Deep-learning Model Secret with GPU Context-switching Side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137. IEEE, 2020.

[YFT20] 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.

[YLX+23] Xiaobei Yan, Xiaoxuan Lou, Guowen Xu, Han Qiu, Shangwei Guo, Chip Hong Chang, and Tianwei Zhang. MERCURY: An Automated Remote Side-channel Attack to Nvidia Deep Learning Accelerator. In *2023 International Conference on Field Programmable Technology (ICFPT)*, pages 188–197. IEEE, 2023.

[YMY+20] 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.

[ZGW+23] Mingyi Zhou, Xiang Gao, Jing Wu, John Grundy, Xiao Chen, Chunyang Chen, and Li Li. ModelObfuscator: Obfuscating Model Information to Protect Deployed ML-based Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1005–1017, 2023.

[ZYC+21] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. Stealing Neural Network Structure Through Remote FPGA Side-Channel Analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.