A Divide-and-Conquer Strategy for Hard-Label Extraction of Deep Neural Networks via Side-Channel Attacks

Benoît Coqueret[2], Mathieu Carbone[2], Olivier Sentieys[3] and Gabriel Zaid[1*]

[1] CryptoExperts, Paris, France, {firstname.lastname}@cryptoexperts.com
[2] Thales ITSEF, Toulouse, France, {firstname.lastname}@thalesgroup.com
[3] University of Rennes, INRIA, IRISA, Rennes, France, {firstname.lastname}@inria.fr

Abstract. During the past decade, Deep Neural Networks (DNNs) have proven their value across a wide variety of applications; however, despite their importance, protecting their intellectual property remains an open issue. Recent work has successfully extracted DNNs using cryptanalytic methods in hard-label settings, showing that it is possible to copy a DNN with high fidelity, *i.e.*, a high degree of similarity in correct/incorrect output predictions which corresponds to the proportion of samples for which the extracted model produces the same prediction as the original model. However, these methods have only been demonstrated on Multi-Layer Perceptrons (MLPs) and are sensitive to non–fully connected layers and special-case neurons. To overcome these limitations, we base our contribution on a divide-and-conquer paradigm. We introduce a new black-box side-channel attack that splits the targeted DNN into several linear components, for which cryptanalytic extraction can be performed. Building on this decomposition, we propose an end-to-end framework specifically designed for hard-label settings, not limited to fully connected layers, and robust to special-case neurons, while improving extraction fidelity. We validate our contribution by successfully extracting all architectures previously targeted in the literature, as well as several new architectures implemented on a microcontroller unit. These include an MLP with 1.7 million parameters, nearly doubling the previous largest number of extracted weights, and a shortened MobileNetv1, which for the first time includes pooling layers and depthwise separable convolutions. Our framework successfully extracts all of these DNNs with high fidelity (88.4% for MobileNetv1 and 93.2% for the MLP). Finally, we use the copied model to generate adversarial examples and achieve near white-box performance on the victim model (95.8% and 96.7% transfer rates).

Keywords: Side-Channel Attacks · Deep Neural Networks · Cryptanalytic extraction · Hard-label

1 Introduction

During the last decade, the number of tasks for which Deep Neural Networks (DNNs) have proven their effectiveness has steadily increased, leading to the widespread adoption of these algorithms in a large variety of fields. From computer vision to text translation, DNNs are now everywhere, and the best models have become valuable intellectual property (IP). At the same time, parallel efforts from hardware designers have made the deployment of DNNs on edge devices possible. However, due to their high value, the IP of deployed models must be protected against new attacks arising from the embedded-system context. Since the

*Work done when the author was at Thales ITSEF.

publication of the first side-channel attack against the IP of an embedded DNN [BBJP19], the number of physical attacks against DNNs has greatly increased. Several methodologies using side-channel attacks with the objective of extracting DNN hyperparameters have been proposed [JMPR23, GQZ+24]. DNN parameters have also been targeted by physical attacks, either via side-channel analysis [YMY+20, JMPR22] or through fault injection [RCYF21, HDMD23]. These types of attacks, which target the parameters of the model, aim to copy the targeted model and thus perform a model extraction attack.

Model extraction is a threat to any deployed DNN, and as such there exists a large variety of methods—and even objectives—for these attacks. We can characterize the two main types of objectives, or adversarial goals, for model extraction using the terminology introduced in [JCB+19]: *accuracy-based model extraction* and *fidelity-based model extraction*. The former aims to obtain a substitute model with good performance on the task of the targeted model without performing the full training process. The latter aims to retrieve the exact internal parameters (*e.g.*, trainable weights) of the targeted model in order to obtain a copy as close as possible to the original. The copied model can then be used to gain information about the victim's DNN and potentially mount more powerful attacks against it. In this study, we consider only *fidelity-based model extraction*.

Cryptanalytic-based extraction. Jagielski *et al.* [JCB+19] were the first to propose a functional methodology for fidelity-based model extraction of single-layer neural networks (NNs) using the ReLU (Rectified Linear Unit) activation function. They exploit the gradient of the DNN to gain access to what they define as critical points. These points correspond to inputs for which the activation value of a specific neuron is zero. Such points are observable in DNNs using ReLU activations, as they correspond to locations where discontinuities occur in the gradient of the network. Using this observation, they are able to extract a single-layer fully connected NN using a least-squares (LSTSQ) algorithm. Both [CJM20] and [RK19] successfully extend this result to deeper architectures, with the method proposed in [CJM20] achieving the highest fidelity to date. One key limitation of the method proposed in [CJM20] is the extraction of neuron signs, which relies on an exhaustive search. This information is critical, as assigning an incorrect sign leads to an unsuccessful attack (*i.e.*, activated neurons become deactivated, and vice versa). Canales-Martínez *et al.* argue that this approach does not scale well to larger architectures [CMCSH+24]. They instead use crafted perturbations that activate specific neurons to infer this information [CMCSH+24]. While effective, all these methods rely on access to the full confidence-score output vector of the DNN, which enables gradient estimation. This represents an idealized attacker setting, and recent works have aimed to remove this assumption in order to move toward more realistic scenarios. Chen *et al.* [CDG+25] recently demonstrated that it is possible to extract DNNs in hard-label settings for small networks. This result was further extended by Carlini *et al.* [CCSH+25], who successfully extracted a DNN with four hidden layers using only hard-label outputs. However, although these works no longer require confidence scores, their designs remain sensitive to non–fully connected layers and special-case neurons, as in previous methods [CJM20, CMCSH+24]. This intuition was confirmed by Ito *et al.* [IMT25], who showed that special-case neurons require an exponential number of queries to extract as network depth increases. Consequently, in practice, the assumptions required to extract DNNs in hard-label settings within polynomial time do not hold [IMT25].

In parallel, several fidelity-based model extraction methods have been proposed using physical attacks. In [RCYF21, HDMD23], fault injection is used to recover a subset of the weight bits, while constraint learning is applied to infer the remaining parameters. Unlike the previously mentioned methods, these approaches are not restricted to fully connected DNNs and have successfully extracted complex architectures, such as ResNet-34 or VGG-11 [RCYF21]. However, these methods are either limited to DRAM-based inference platforms,

such as Machine-Learning-as-a-Service (MLaaS) systems [RCYF21], or require access to an open device over which the attacker has full control, *i.e.* a white-box setting—to locate the memory regions storing the weights [HDMD23]. To the best of our knowledge, there is currently no method capable of extracting complex architectures, *i.e.* not restricted to fully connected layers, without requiring access to confidence scores or an open device.

Contributions. To overcome these issues, we present the first framework that combines side-channel attacks with cryptanalytic-based extraction of DNNs in hard-label settings. To do so, we introduce a new generic methodology that subdivides the extraction problem into a divide-and-conquer strategy. This approach offers higher precision than previous frameworks for weight extraction, improving the extraction of the targeted model while operating in a more restricted context. Furthermore, since our methodology is based on a different paradigm than previous ones, it is not impacted by the previously mentioned limitations, *i.e.*, the extraction of special-case neurons, restrictions on layer types, and, most importantly, hard-label settings.

We summarize the major contributions of our work as follows:

- We highlight the limitations of extracting non-fully connected layers when state-of-the-art cryptanalytic-based methods are used in hard-label settings.
- We introduce a new end-to-end methodology based on a side-channel oracle that is more robust to special-case neurons and non-fully connected layers, thus enabling the extraction of complex DNNs.
- To assess the suitability of the side-channel oracle, we apply a clustering-based side-channel attack to a state-of-the-art implementation of a constant-time ReLU activation function [MBC21].
- Finally, we validate our methodology by extracting all architectures targeted by state-of-the-art frameworks using 32- and 64-bit data, and by targeting two new architectures: an MLP with more than 1.7 million parameters and a shortened version of MobileNetv1 embedded on an STM32F767ZI using the X-Cube-AI framework. This leads to the first cryptanalytic-based extraction of depthwise separable convolution, batch normalization, and pooling layers.

We provide an implementation of our code at https://github.com/bcoqueret/Side_channel_cryptanalytic_extraction_of_DNN.

Threat Model. In the remainder of this work, we consider an attacker whose objective is to extract the weights of a DNN deployed on an embedded device, such as an MCU, in a hard-label setting. Following prior work [JCB+19, RK19, CJM20], we restrict our analysis to Deep ReLU networks and assume that the architecture of the target model is known to the attacker. We do not, however, restrict the DNN to fully connected layers.

As we consider a side-channel setting, we assume that the attacker has physical access to the device executing the target DNN, enabling the acquisition of side-channel measurements (*e.g.*, electromagnetic (EM) traces in our case) during inference. In contrast to existing works (*e.g.*, [ZDF23, HDMD23]), we consider an attacker using non-profiled attacks based on unsupervised learning methods. Consequently, the attacker has no *a priori* knowledge of the model parameters and does not have access to the training or test datasets used to train the target DNN.

We summarize in Table 1 the state-of-the-art (SOTA) frameworks performing *fidelity-based model extraction* using cryptanalytic methods, fault injection, or side-channel attacks, as well as our contribution.

Table 1: Overview of the state-of-the-art of fidelity-based model extraction attacks.

Approach	Attack type	Full extraction (weight + bias)	Hard-label setting[1]	Applicable to non-fully connected DNNs	Random queries[2]	DNN's datatype tested	Targeted architecture (Most complex)
[RK19]	Cryptanalytic	✓	✗	✗	✓	64-bit	MLP 10-20-20-1
[CJM20]	Cryptanalytic	✓	✗	✗	✓	64-bit	MLP 40-20-10-10-1
[CDG+25]	Cryptanalytic	✓	✓	✗	✓	64-bit	MLP 1024-2-2-1
[CCSH+25]	Cryptanalytic	✓	✓	✗	✓	64-bit	MLP 3072-256×3-64-10
[BBJP19]	Side-channel	✗	✓	✓	✓	32-bit	MLP 784-200×4-10
[ZDF23]	Side-channel	✓	✗	✗	✓	64-bit	LeNet5
[RCYF21]	Fault Injection	✓	✓	✓	✗	8-bit	ResNet34 and VGG11
[HDMD23]	Fault Injection	✓	✗	✓	✗	8-bit	CNN 3×(Conv + Pooling + ReLU)-Linear
This work	Cryptanlytic and side-channel	✓	✓	✓	✓	32- and 64-bit	Shortened MobileNetv1 MLP 3072-256×3-64-10

[1] Attacks which are **not** in hard-label settings suppose access to the confidence scores.
[2] Contributions which are **not** using random queries use a subset of the training or a testing dataset to perform part of their attacks.

2 Background

2.1 Notations

Let calligraphic letters $\mathcal{X}$ denote sets, the corresponding capital letters X (resp. bold capital letters) denote random variables (resp. random vectors $\mathbf{T}$), and the lowercase x (resp. $\mathbf{t}$) denote their realizations. We will use $\mathbf{T}_i$ to describe the i-th element of a vector $\mathbf{T}$. Throughout this paper, the function modeled by a DNN is denoted as $f : \mathcal{X} \to |\mathcal{Y}|$, which characterizes its ability to classify a data $X \in \mathcal{X}$, *e.g.*, an image, over a set of $|\mathcal{Y}|$ classes. A DNN characterized by the function f and the set of weights θ is denoted f_θ, and we denote $f_{\hat{\theta}}$ the model extracted from f_θ. The probability of observing an event X is denoted by $\texttt{Pr}[X]$. Finally, we denote $\mathcal{D}_{\mathcal{X}}$, the distribution over the set of data $\mathcal{X}$.

2.2 Fidelity-based model extraction

Model extraction attacks target the confidentiality of a deployed model by attempting to obtain a copy of the victim's model. The most common objective is to use the copied model to benefit from the target's capabilities, *e.g.*, commercial value or performance on specific tasks, without having to train a model. Another possibility is to use the copied model to gain information about the target and mount higher-level attacks against it, such as the generation of adversarial examples in a white-box setting [SZS+14, CW16] or membership inference attacks [FJR15]. This diversity of attack scenarios leads to several adversarial goals in model extraction. Following the notation introduced in [JCB+19], we consider two main goals, namely *task accuracy* and *fidelity*.

The first goal aims to extract a model $f_{\hat{\theta}}$ such that, for the true task distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$ over the input space $\mathcal{X}$ and label space $\mathcal{Y}$, $f_{\hat{\theta}}$ maximizes $\texttt{Pr}_{X,Y}[\texttt{argmax}(f_{\hat{\theta}}(X)) = Y]$. In practice, this can be achieved through *learning-based* methods and optimization. However, this approach introduces variability in the optimization process, which can lead to very different solutions, *i.e.* models, due to convergence to different local minima. As a result, this method offers no guarantee from a *fidelity* perspective and is therefore suited for the objective of *task accuracy*.

The second adversarial goal consists of maximizing the similarity between the copied and the original models in their predictions. The strongest form of this attack is *functional equivalence*, which corresponds to obtaining an exact copy of the targeted DNN. In this paper, we focus exclusively on *fidelity-based model extraction*. To evaluate the success of

such extraction, we use the notion of (ϵ, δ)-functional equivalence.

Definition 1 ((ϵ, δ)-functional equivalence [CJM20])**.** Two models f_θ and g_γ are (ϵ, δ)-functionally equivalent on $\mathcal{X}$ if:

$$\texttt{Pr}_{X \in \mathcal{X}}[|f_\theta(X) - g_\gamma(X)| \leq \epsilon] \geq 1 - \delta.$$

The advantages of *fidelity-based model extraction* have motivated extensive research in this area. In particular, side-channel attacks [BBJP19, JMPR22, ZDF23, HCW+25] and fault-based attacks [RCYF21, HDMD23] have been used to target either all weights of a DNN individually or partial information about them.

2.3 Side-channel attacks

Historically, side-channel attacks (SCAs) are a class of cryptographic attacks in which an adversary exploits implementation-level vulnerabilities of real-world cryptosystems by analyzing physical leakages, such as power consumption or electromagnetic (EM) emissions. During cryptographic execution, side-channel traces correlated with key-dependent intermediate variables are recorded. Key discrimination is then achieved by applying statistical distinguishers to a large number of traces across all key hypotheses. In [BBJP19], Batina *et al.* were among the first to adapt these techniques to DNNs, targeting both architecture and parameters using Pearson correlation in a Correlation Electromagnetic Analysis (CEMA) combined with a Simple Power Analysis (SPA). They specifically target the Hamming weight of the accumulated sum between a neuron's weight vector and its input vector on ARM Cortex-M3 and ATmega328P devices. However, limitations related to the ReLU function, error propagation, and bias extraction were identified in [JMPR22]. Since then, CEMA has been extended to extract DNN weights on microcontrollers [JMPR22] and GPUs [HCW+25]. Several other side-channel techniques originally developed for cryptographic systems have also been adapted to DNNs. In particular, deep learning–based side-channel attacks [MPP16, CCP17] have been successfully applied to extract DNN hyperparameters. Gao *et al.* [GQZ+24] trained meta-models on physical traces to recover model architectures with high precision. Earlier, Maia *et al.* [MXL+21] reconstructed ResNet and VGG architectures running on GPUs using physical traces. All these attacks rely on supervised learning to construct the distinguisher, which requires access to an open device providing ground-truth labels. This assumption is restrictive and motivates the exploration of more permissive threat models based on unsupervised learning, as specified in our threat model (see Section 1).

These approaches thus exhibit limitations in terms of attack complexity, supported architectures, or threat model assumptions (see Table 1). On the other hand, cryptanalytic-based attacks have proven effective against shallow networks [MSDH18, JCB+19] and were later extended to deeper architectures [RK19, CJM20]. These methods are briefly introduced in the following section.

2.4 Cryptanalytic extraction methodology

Both methods in [RK19, CJM20] extract DNN weights by iteratively targeting each neuron and monitoring its state in order to identify its critical points and the associated neuron-induced hyperplane. These concepts are defined as follows.

Definition 2 (Critical point and state of a neuron [CJM20])**.** Given an input $X \in \mathcal{X}$, let $V(\eta; X)$ be the function characterizing the input of the neuron η before applying the activation function. Then X is said to be a critical point to the neuron η if $V(\eta; X) = 0$. If $V(\eta; X) > 0$ (resp. $V(\eta; X) < 0$) then η is said to be active (resp. inactive).

Definition 3 (Neuron-induced Hyperplane)**.** The hyperplane of a neuron η corresponds to the set $\mathcal{H}_\eta \subseteq \mathcal{X}$ where $\forall X \in \mathcal{H}_\eta, V(\eta; X) = 0$.

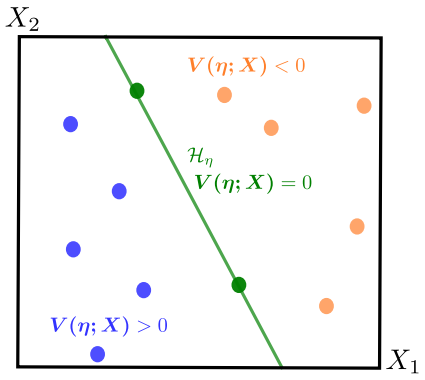


Figure 1: Neuron-induced hyperplane.

An example of critical points, neuron states, and the neuron-induced hyperplane is shown in Figure 1. Identifying critical points is of high importance, as it allows the determination of the neuron-induced hyperplane equation. This equation can then be used to recover the neuron's weights with high precision:

$$\sum_{i=0}^{n} \boldsymbol{\theta}_{\eta,i} \boldsymbol{X}_i + \beta_\eta = 0, \boldsymbol{X} \in \mathbb{R}^{n+1}, \tag{1}$$

with $\boldsymbol{\theta_\eta}$ (resp. β_η) the weight vector (resp. the bias) associated with the neuron η. It is important to note that for a neuron η with $n+1$ parameters, the induced hyperplane is an n-dimensional piecewise-linear surface [HR19]. Thus, from Equation 1, we obtain,

$$\boldsymbol{X}_0 = \sum_{i=1}^{n} \frac{-\boldsymbol{\theta}_{\eta,i}}{\boldsymbol{\theta}_{\eta,0}} X_i - \frac{\beta_\eta}{\boldsymbol{\theta}_{\eta,0}}, \boldsymbol{\theta}_{\eta,0} \neq 0. \tag{2}$$

After extracting a critical point, the neuron's weights can be recovered up to a *scaling factor* by sampling along the standard basis vectors of $\mathcal{X}$. This process is referred to as *signature search* in [CJM20].

Definition 4 (Neuron signature [CMCSH+24])**.** The signature of a neuron η corresponds to a vector $\boldsymbol{S_\eta} \in \mathbb{R}^{n+1}$, for which there exists a scalar $\alpha \in \mathbb{R}$ such that $\boldsymbol{\theta_\eta} = \alpha \cdot \boldsymbol{S_\eta}$.

In Equation 2, the vector $\boldsymbol{S_\eta} = (1, \frac{\boldsymbol{\theta}_{\eta,1}}{\boldsymbol{\theta}_{\eta,0}}, \ldots, \frac{\boldsymbol{\theta}_{\eta,n}}{\boldsymbol{\theta}_{\eta,0}})$ is a signature of the neuron η and $\boldsymbol{\theta}_{\eta,0}$ is the scaling factor. Since the ReLU function is equivariant under positive scaling, *i.e.*, $\forall x \in \mathbb{R}$ and $\forall c \in \mathbb{R}^+$, $\texttt{ReLU}(c \cdot x) = c \cdot \texttt{ReLU}(x)$, extracting neuron signatures rather than exact weight vectors allows faithful reconstruction. As long as the sign of the ReLU input is preserved, activation values can be re-scaled in subsequent layers. Thus, the precise value of the scaling factor is irrelevant; only its sign matters. Consequently, this approach recovers a functionally equivalent model rather than an exact copy.

Under these assumptions, model extraction proceeds iteratively for each neuron in two steps:

1. Extraction of the neuron's signature.
2. Extraction of the sign of the scaling factor to ensure that the neuron's state remains the same.

These steps are detailed in Subsection 3.1. While these methods have successfully enabled complete extraction of Deep-ReLU networks, significant challenges remain, such as the full extraction of non–fully connected DNNs in hard-label settings (see Table 1). These limitations motivate the development of a new methodology that combines cryptanalytic extraction with side-channel analysis.

3 Motivations and extension using side-channel attacks

Although the methods presented in the previous section demonstrate strong results, *fidelity-based model extraction* remains a difficult task. First, in Subsection 3.1, we provide a recap of the different steps required to perform cryptanalytic-based extraction attacks. Then, in Subsection 3.2, we present inherent difficulties that have not been addressed in these attacks, *e.g.* restrictions on the targeted architecture, the high precision required to estimate gradients, and special cases of neurons. Finally, in Subsection 3.3, we describe a divide-and-conquer methodology using a side-channel oracle to tackle the problems identified in Subsection 3.2.

3.1 Historical methodology for cryptanalytic-based extraction

As introduced in Subsection 2.4, multiple steps are required to conduct a cryptanalytic-based extraction attack. First, an attacker must retrieve each neuron's signature by identifying a set of critical points. This step requires determining the neuron's state and, consequently, extracting the corresponding signature. Then, given a set of signatures, the attacker must identify the sign of the scaling factor of each neuron in order to correctly perform fidelity-based model extraction. This section provides an overview of these steps and highlights the most recent state-of-the-art contributions extending the seminal work.

Extraction of the neuron's signature. By considering the targeted model $f_{\boldsymbol{\theta}}$ as an oracle, the search for a neuron's signature can be performed directly using its input–output pairs. Indeed, if the targeted model is composed of piecewise-linear activation functions, then $f_{\boldsymbol{\theta}}$ is also piecewise linear, and discontinuities can be observed in the gradient of $f_{\boldsymbol{\theta}}$. These discontinuities correspond to changes in the activation state of neurons within the DNN. More specifically, the discontinuity in the gradient of the ReLU function is localized at 0, which implies that by monitoring the gradient of the targeted model, critical points for each neuron can be identified. However, performing signature extraction using these discontinuities remains challenging and requires strong assumptions. First, the target model's output must expose either confidence scores or logits, enabling gradient estimation through finite differences. Second, since it is impossible to determine which neuron corresponds to a given critical point, a large number of critical points must be sampled to ensure that each neuron is represented. Third, in deeper layers, on average, half of the neurons are deactivated [CJM20], which implies that many entries of a critical point are zero and therefore prevent recovery of the associated weights. Carlini *et al.* [CJM20] address this issue by recombining signatures obtained from different critical points. However, this requires ensuring that the signatures correspond to the same neuron and layer. To mitigate this, they introduce a consistency check based on the assumption that signatures from different neurons or layers lead to inconsistent results. Finally, the model output must be returned with high numerical precision to ensure accurate gradient estimation. In [JCB+19, CJM20], the authors successfully extract ReLU-based networks operating with double-precision (*i.e.*, 64-bit floating-point) arithmetic. While neuron signatures can be revealed through gradient discontinuities, the sign of the scaling factor is not directly accessible to the attacker. Consequently, an additional extraction step is required.

Extraction of the neuron's sign. Carlini *et al.* [CJM20] extract the sign of the neurons by exhaustively testing the 2^m possible sign configurations, where m denotes the number of neurons in the layer. As this approach quickly becomes impractical for large DNNs, Canales-Martínez *et al.* [CMCSH+24] propose an alternative method based on crafting inputs that selectively activate specific neurons to infer their sign. Their approach successfully extracts the signs of neurons in a model with eight hidden layers, each containing 256 neurons. This process is applied sequentially (*i.e.*, for each neuron in each layer), starting from the first layer and proceeding toward the last. As a result, this attack operates in a gray-box setting, assuming knowledge of the architecture and access to confidence scores.

New directions in the SOTA. Chen *et al.* [CDG+25] were the first to propose a cryptanalytic extraction method for DNNs operating in hard-label settings, targeting binary classifiers. Their approach relies on *decision boundary points*, *i.e.*, inputs located on the boundary between two labels, and was demonstrated on DNNs with two hidden layers. However, this method exhibits exponential time complexity [CCSH+25] and is limited to binary classification, motivating the search for alternative approaches. Carlini *et al.* [CCSH+25] introduce such an alternative by leveraging decision boundary points[1] to identify critical points. Due to the piecewise-linearity of ReLU networks, the input space is partitioned into linear regions, referred to as *cells*. In prior methods, boundaries between cells, corresponding to neuron-induced hyperplanes, were identified through gradient estimation, which is not feasible in hard-label settings. The key insight of [CCSH+25] is that decision boundaries must be flat hyperplanes within a given cell. Thus, they identify *dual points*, defined as points that lie both on a decision boundary and on a cell boundary, by analyzing the geometry of the decision boundary. An illustration of such points is shown in Figure 2a. Since these points are also critical, they can be used to extract neuron signatures (see Definition 4). However, this strategy presents several limitations, which are discussed in Subsection 3.2.

Model extraction in hard-label settings is not the only recent direction explored in the SOTA. Indeed, since the last layer of a DNN typically does not use a ReLU activation function, its extraction requires a different strategy. In soft-label settings, attackers can exploit confidence scores as labels for supervised learning [JCB+19]. In contrast, hard labels do not provide sufficient information to guarantee functional equivalence, and extraction of the last layer was not addressed in [CCSH+25]. Recently, Canales *et al.* [CMS25] proposed a first methodology to address this issue. Their approach uses points on the decision boundary to form a system of equations that enables extraction of the final layer's parameters. However, such systems are not always full rank. To address this, the authors fix a subset of variables, allowing recovery of an equivalent model. Although this method does not guarantee exact parameter recovery, the extracted model achieves perfect fidelity with respect to the target [CMS25].

While these developments improve the practicality of cryptanalytic-based extraction, several open limitations remain, which are discussed in the following section.

3.2 Natural difficulties in cryptanalytical extraction of DNNs

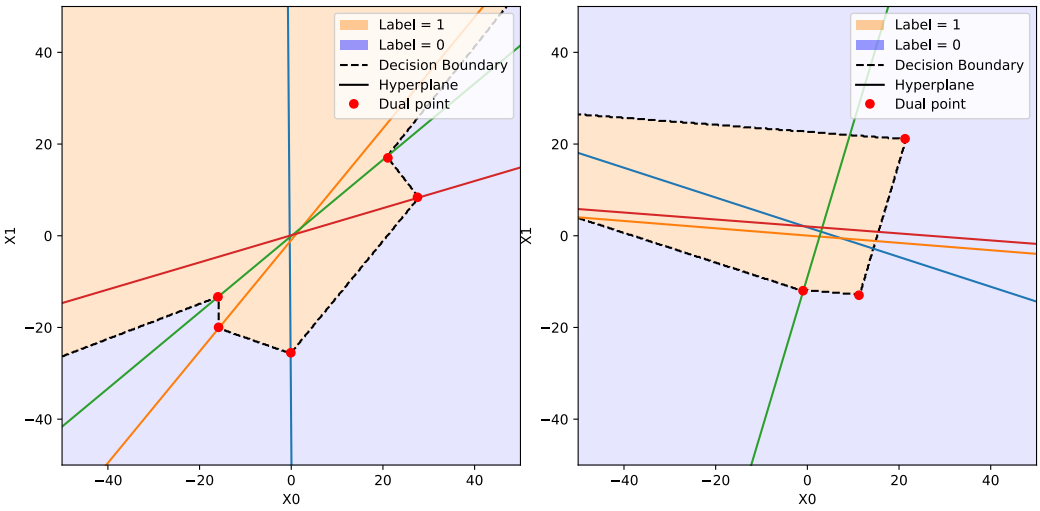
Due to the complexity of cryptanalytical extraction, several limitations persist. The most direct limitations of the methodologies presented in [JCB+19, CJM20] are those related to the threat model. On the first hand, gradient estimation requires access to confidence scores or logits with high numerical precision (*i.e.*, 64-bit floating-point values). On the other hand, no prior work has successfully demonstrated extraction of DNNs that include

[1] The authors in [CCSH+25] refer to these points as transition points. Out of clarity we choose to use the notation decision point in this work.

non-fully connected layers, such as max or average pooling layers. While the first limitation appears natural, the second requires further explanation.

Limitation to extract non-fully connected layers. As mentioned in Subsection 3.1, the most promising methodology in hard-label settings was introduced in [CCSH+25]. In this work, Carlini *et al.* propose using the decision boundary to identify dual points, *i.e.* points that lie both on the decision boundary and on a neuron-induced hyperplane, to extract trained parameters, namely weights and biases. They successfully demonstrate extraction of all hidden layers of a four-layer DNN trained on CIFAR-10 using only hard labels. However, their evaluation is limited to fully connected networks. We argue that this methodology is not directly applicable to more complex architectures, especially those including non-fully connected layers such as max-pooling layers.

To illustrate this limitation, we train a simple linear neural network with one hidden layer on a binary classification task. In Figure 2a, we observe that the shape of the decision boundary changes only when crossing a neuron-induced hyperplane, which is the core assumption of the method in [CCSH+25]. In this case, the identified points effectively correspond to dual points, and the hyperplane can be recovered in hard-label settings. However, when training the same network with the addition of a max-pooling layer, we obtain comparable classification performance, yet observe in Figure 2b that the decision boundary no longer correlates with the neuron-induced hyperplanes. This discrepancy



(a) Fully connected network.

(b) DNN with a max pooling layer.

Figure 2: Illustration of the concept of dual points as introduced in [CCSH+25] on a MLP with one hidden layer of four neurons each associated with a different hyperplane represented by the colored lines.

leads to incorrect identification of dual points and, consequently, incorrect estimation of weights. Therefore, in such configurations, the approach proposed in [CCSH+25] is no longer applicable, and cryptanalytical model extraction fails. This behavior can be explained by the fact that while a change of linear cell may occur, the max-pooling operation can suppress this change if the corresponding activation is not selected by the pooling layer. As a result, the change is not visible in the decision boundary. Hence, the attacker cannot correctly identify dual points, leading to poor extraction results. While historical approaches do not scale well to non-fully connected layers, additional limitations also arise in more favorable settings, *i.e.* when targeting fully connected layers.

Limitation to extract fully connected layers. Even when extracting fully connected layers, which represents an ideal scenario for the attacker, cryptanalytic extraction using the methodology described in Subsection 3.1 remains challenging for several reasons. We highlight three issues that are addressed in Subsection 3.3.

A first issue, reported in [CMCSH+24], concerns special cases of neurons that almost never change state. Two types of such neurons are identified:

- ***Always-Off.*** These correspond to *dead* neurons whose parameters cause their pre-activation value to be negative for most inputs. In ReLU-based DNNs, their output is therefore always zero.
- ***Always-On.*** These neurons exhibit the opposite behavior, with pre-activation values that are always positive. They therefore behave as linear transformations.

For these neurons, the number of queries required to identify critical points can exceed that of typical neurons by an order of magnitude (see Subsection 6.2). Despite their practical importance, no prior work provides a systematic solution for handling these cases.

A second issue is that input–output pairs do not allow direct identification of the neuron associated with a given critical point. Consequently, attackers cannot verify whether all neurons have been successfully extracted (*i.e.* all neurons have critical points). Carlini *et al.* [CJM20] address this by sampling a large number of critical points relative to the number of neurons, ensuring coverage with high probability. However, this leads to redundant queries and significantly increases attack complexity, particularly for deep architectures.

Finally, the absence of direct mapping of the critical points to the targeted neuron leads to another problem brought to light in [LSE+25]. More specifically, their concern is that the consistency check used by Carlini *et al.* in [CJM20] to aggregate the incomplete signature of a given neuron is not correct. In deeper layers, ReLU deactivations often produce incomplete signatures because some input components are suppressed. Since extracting a neuron requires a complete signature, Carlini et al. [CJM20] propose aggregating multiple partial signatures to recover missing entries. Their method merges signatures by verifying that their shared entries are proportional, increasing the aggregated signature size with each merge. They argue that aggregated signatures of size greater than two must originate from the targeted layer, as deeper-layer signatures are unlikely to satisfy the merge condition. However, Liu et al. show that signatures from deeper layers can also be merged when neurons share identical activation patterns across intermediate layers, causing the consistency check to fail and resulting in incorrect signature recovery. Several solutions to mitigate this issue have been introduced in [LSE+25], which have led to the successful extraction of signatures from deeper layers. More particularly, they are able to increase the SOTA by extracting 95% of the weights in each layer of DNN with eight hidden layers on MNIST, while previous methodologies could not perform the extraction beyond the third layer [LSE+25]. However, even with their method, targeting very deep architectures remains difficult. Furthermore, while these improvements make the attacks more practical, they do not change the threat model and still consider full access to the confidence scores.

This section highlights the need for a new methodology for cryptanalytic-based model extraction. In the following section, we introduce such a methodology by adopting a divide-and-conquer strategy.

3.3 Divide-and-conquer methodology

This section provides a high-level overview of the proposed approach, while Section 4 presents a detailed description of each step. Our objective is to introduce an alternative fidelity-based model extraction methodology that is robust to special-case neurons and

capable of extracting deep networks composed of non-fully connected layers in hard-label settings. We argue that many of the existing limitations arise from targeting the model as a whole and do not apply to single-layer neural networks (NNs). To facilitate cryptanalysis-based extraction attacks in hard-label settings, and thereby mitigate the limitations discussed in Subsection 3.1 and Subsection 3.2, we propose an approach based on a side-channel oracle O that decomposes the problem into the extraction of successive single-layer NNs. Accordingly, a DNN with L layers is decomposed into L one-layer NNs, where the input to the ℓ-th network is the output of the $(\ell-1)$-th network, for $\ell \in [1, L]$. This divide-and-conquer strategy reduces the attack complexity by enabling the analysis of the DNN's internal variables rather than treating the network as a monolithic system. A high-level illustration of the extraction pipeline is shown in Figure 3. The remainder of this section focuses on the extraction of a single-layer NN; extracting a DNN with L layers then requires repeating this process $L-1$ additional times.

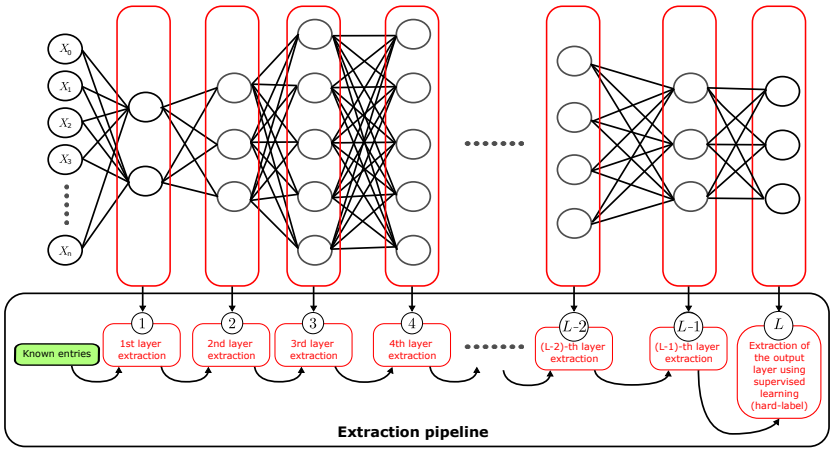


Figure 3: High-level description of the divide-and-conquer approach when targeting a L hidden layers DNN.

As specified in Subsection 3.1, cryptanalytic-based extraction requires two steps to extract one neuron of the ℓ-th layer: one for extracting the neuron's signature and one for determining the sign of the scaling factor. As a reminder, extracting a neuron's signature requires identifying critical points for each neuron of the ℓ-th layer. To perform this identification, we use a side-channel oracle O, which returns a boolean value equal to *True* if two neurons are associated with the same state for two given inputs. We justify this choice by the fact that activation functions of DNNs have already been targeted using distinguishers exploiting timing differences in [BBJP19, MBC21], leading to the identification of the activation function and even the weights' mantissa. While effective, both previously mentioned approaches rely on supervised learning and therefore require access to an open device to train the distinguisher. In this work, we consider a less restrictive adversary who does not require such an open device to conduct the attack (see Table 1). Consequently, a supervised learning strategy is not applicable. To address this limitation, we propose an alternative distinguisher based on an unsupervised *k-means* clustering algorithm [DHG01]. The idea is to generate a sufficient number of EM traces using random inputs to build a dataset in which both classes, *i.e.*, active and inactive neurons, are evenly represented. We then apply the *k-means* algorithm to identify two clusters corresponding to the two neuron states. As mentioned previously (see Section 4), it is not initially necessary to determine which cluster corresponds to the active or inactive state, since this information is not required to identify a critical point. These clusters are

then used as references to infer the state of a neuron. Finally, given the EM trace of a new neuron, we use the *Euclidean* distance to the centroids of the two clusters as a distinguisher to determine the neuron's state. Under the assumption that the implementation of the ReLU function is identical for all neurons across all layers, this distinguisher can be used as an oracle to perform the complete extraction.

An additional advantage of such a side-channel oracle is the direct mapping from a critical point to its corresponding neuron. Indeed, as discussed in Subsection 3.2, a key limitation of classical cryptanalytic methods is their inability to determine which neuron corresponds to a given critical point, resulting in the extraction of unnecessary critical points. This limitation is not present in our methodology, as the oracle directly provides the state of the targeted neuron. We are therefore able to localize which neuron corresponds to each identified critical point. Furthermore, this direct mapping naturally eliminates the issues related to merging signatures described in [LSE+25].

Given this high-level description of the divide-and-conquer strategy, we summarize the extraction of each neuron of the ℓ-th layer into three stages. This strategy is illustrated in Figure 4:

- During the first stage of our methodology, we use a side-channel-based oracle to perform a binary search to identify critical points, following Algorithm 1. In contrast to historical approaches (*e.g.* [JCB+19, CJM20]), this contribution offers two main benefits: first, it does not require access to confidence scores or logits to identify critical points; second, the oracle can also be used to extract non-fully connected layers such as pooling layers. This stage is detailed in Subsection 4.1 and discussed in Subsection 4.4.

- The second stage consists of approximating the equation describing the hyperplane induced by the targeted neuron. The critical points identified in Stage 1 are used following the method introduced in Subsection 4.2. This stage also enables, for the first time, the extraction of special-case neurons (*i.e.*, always-off, always-on, and input-off neurons).

- In the third stage, we propose a new approach to extract the sign of each neuron by leveraging the side-channel oracle. Described in Subsection 4.3, this approach allows the extraction of the sign of each neuron in the ℓ-th layer without requiring additional queries to the targeted DNN.

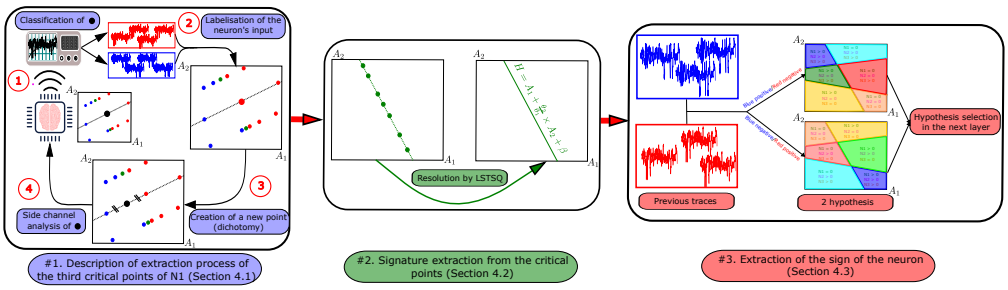


Figure 4: Description of our complete methodology for extracting a neuron included into the ℓ-th layer.

Once the three stages have been performed for the $(L-1)$ one-layer NNs, the final layer must be extracted. As specified in Subsection 3.1, the extraction process for the

last layer differs in hard-label settings. While Canales *et al.* [CMS25] use points on the decision boundary to extract the final trainable parameters, we instead employ supervised learning on a dataset composed of the random inputs generated for the previous layers and their corresponding hard labels. This dataset is used to train the last layer to behave similarly to that of the targeted model. Although this methodology enables extraction of the final layer in hard-label settings, a discussion is provided in Subsection 6.3 to evaluate the impact of this strategy.

4 Subdividing DNN to improve cryptanalytical extraction

In this section, we detail the three-stage methodology introduced in Subsection 3.3. In Subsection 4.1, we describe how to use a side-channel oracle function to identify critical points. Then, in Subsection 4.2, we illustrate how this set of critical points can be used to extract the parameters of each layer and introduce a new strategy to handle special-case neurons. Finally, in Subsection 4.3, we present a method to extract the signs of all scaling factors within a layer using a single hypothesis.

4.1 Extraction of critical points

In our methodology, we assume access to a side-channel oracle $\mathsf{O} : \mathbb{N}\times\mathbb{N}\times\mathcal{X}\times\mathcal{X} \to \mathcal{B}$, which returns a boolean value indicating whether two inputs $\boldsymbol{X} \in \mathcal{X}$ and $\boldsymbol{X}' \in \mathcal{X}$ are associated with the same state for two neurons $(\eta, \gamma) \in \mathbb{N}^2$, *i.e.*, whether $V(\eta; \boldsymbol{X}) = V(\gamma; \boldsymbol{X}')$ (see Definition 2). As such, the side-channel oracle O returns True when the two inputs correspond to the same state, and False otherwise. Since this function is often applied to different inputs for the same neuron η, we use the following simplified notation when no ambiguity arises: $\mathsf{O}(\eta, \boldsymbol{X}, \boldsymbol{X}') = \mathsf{O}(\eta, \eta, \boldsymbol{X}, \boldsymbol{X}')$. Moreover, throughout this work, we assume that $\mathcal{X} \subseteq \mathbb{R}^{n+1}$, with $n \in \mathbb{N}$. This oracle naturally removes the need for soft labels in the search for critical points.

Remark 1. Note that this oracle does not require knowledge of the true states $V(\eta; \boldsymbol{X})$ and $V(\eta; \boldsymbol{X}')$ associated with the inputs $\boldsymbol{X}$ and $\boldsymbol{X}'$. The only required information is whether $V(\eta; \boldsymbol{X})$ and $V(\eta; \boldsymbol{X}')$ belong to the same state.

To identify critical points, an attacker samples random inputs $\boldsymbol{X}$ and $\boldsymbol{X}'$ in $\mathcal{X}$ until she finds a tuple $(\boldsymbol{X}, \boldsymbol{X}')$ such that $V(\eta; \boldsymbol{X}) \neq V(\eta; \boldsymbol{X}')$. She then updates the tuple $(\boldsymbol{X}, \boldsymbol{X}')$ by performing a binary search using the side-channel oracle O until the distance between $\boldsymbol{X}$ and $\boldsymbol{X}'$ is bounded by a parameter Δ. A high-level description of this procedure is presented in Algorithm 1. Since we perform a binary search, finding a true critical point for the targeted neuron η, *i.e.*, a point $\boldsymbol{X}$ lying exactly on the hyperplane $\mathcal{H}_\eta$ (see Definition 3), is challenging due to numerical approximations. Instead, we introduce a hyperparameter Δ to obtain a point located within a bounded distance from the hyperplane $\mathcal{H}_\eta$.

Under the assumption that the side-channel oracle operates perfectly, this algorithm is equivalent to a dichotomic search over a sorted list of binary values. Since binary search is used to retrieve critical points, the complexity of this phase matches that of [JCB+19]. Specifically, it requires $\mathcal{O}\big(\log_2\big(\frac{L_0}{\Delta}\big)\big)$ queries to obtain a point within distance Δ of the hyperplane, where L_0 denotes the initial distance between the first pair $(\boldsymbol{X}, \boldsymbol{X}')$ such that $\mathsf{O}(\eta, \boldsymbol{X}, \boldsymbol{X}')$ is False. With this side-channel oracle, it is therefore possible to search for all critical points within a layer. Once a set of critical points has been extracted, the attacker must identify and extract the corresponding signatures to recover the hyperplane $\mathcal{H}$.

Algorithm 1 Binary search of a critical point using an oracle.

Require: a side-channel oracle $\mathsf{O} : \mathbb{N} \times \mathbb{N} \times \mathcal{X} \times \mathcal{X} \to \mathcal{B}$, the targeted neuron $\eta \in \mathbb{N}$ and a distance precision $\Delta \in \mathbb{R}$
Ensure: $\boldsymbol{X} \in \mathcal{H}_\eta$
  $(\boldsymbol{X}, \boldsymbol{X}') \leftarrow (\texttt{random}(\mathcal{X}), \texttt{random}(\mathcal{X}))$ ▷ Generate a random tuple in $\mathcal{X}$
  $S \leftarrow O(\eta, \boldsymbol{X}, \boldsymbol{X}')$
  while S is True **do**
    $\boldsymbol{X}' \leftarrow \texttt{random}(\mathcal{X})$
    $S \leftarrow O(\eta, \boldsymbol{X}, \boldsymbol{X}')$
  end while
  while $||\boldsymbol{X} - \boldsymbol{X}'||_2 > \Delta$ **do**
    $\boldsymbol{M} \leftarrow \frac{\boldsymbol{X}+\boldsymbol{X}'}{2}$
    $S_M \leftarrow O(\eta, \boldsymbol{X}, \boldsymbol{M})$
    if S_M is True **then** ▷ Implies that $V(\eta; \boldsymbol{M}) = V(\eta; \boldsymbol{X})$
      $\boldsymbol{X} \leftarrow \boldsymbol{M}$
    else ▷ Implies that $V(\eta; \boldsymbol{M}) = V(\eta; \boldsymbol{X}')$
      $\boldsymbol{X}' \leftarrow \boldsymbol{M}$
    end if
  end while
  return $\boldsymbol{X}$

4.2 Improved signature extraction

Since each critical point provides an equation linking a known input[2] to an unknown weight vector, it is straightforward to construct a system of equations to address this problem.

Generic method. We infer the equation of the neuron-induced hyperplane (see Definition 3) by constructing a linear system based on Equation 1 using the critical points obtained from Algorithm 1. Such a system can then be used to extract the weights. Given a set of $n+1$ critical points $\{\boldsymbol{X}^1, \boldsymbol{X}^2, \ldots, \boldsymbol{X}^{n+1}\}$ such that $\boldsymbol{X}^i \in \mathbb{R}^{n+1}$, the following system of equations can be used to extract the n weights and the bias of the targeted neuron η:

$$\begin{cases} \displaystyle\sum_{i=1}^{n} \frac{-\boldsymbol{\theta}_{\eta,i}}{\boldsymbol{\theta}_{\eta,0}} \boldsymbol{X}_i^1 - \frac{\beta_\eta}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^1 \\ \qquad\qquad \vdots \\ \displaystyle\sum_{i=1}^{n} \frac{-\boldsymbol{\theta}_{\eta,i}}{\boldsymbol{\theta}_{\eta,0}} \boldsymbol{X}_i^{n+1} - \frac{\beta_\eta}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^{n+1} \end{cases} \tag{1}$$

Once a sufficient number of critical points has been sampled using Algorithm 1 to construct a full-rank system, we recover the equation of the hyperplane by solving the corresponding LSTSQ problem. We then extract the signature (see Definition 4) of the targeted neuron by arbitrarily fixing the scaling factor $\theta_{\eta,0}$ to ± 1. The sign of this scaling factor determines the state of the neuron and is therefore a critical parameter to extract. The extraction of the scaling factor sign is detailed in Subsection 4.3.

Remark 2. It is important to note that the presence of linear transformations between the matrix multiplication and the activation function, *e.g.* batch normalization, does not impact the proposed method. Indeed, all such linear operations can be merged into a

[2] This holds since, to extract the ℓ-th layer, it is assumed that the $(\ell-1)$-th layer has already been successfully extracted.

single equivalent transformation, from which an aggregate set of weights can be extracted. This merging process is illustrated in Figure 5.

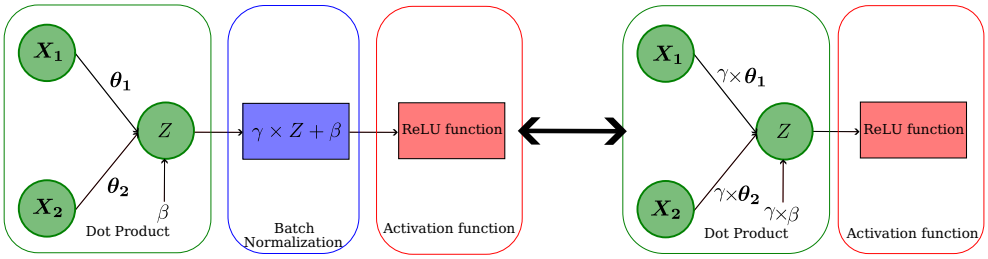


Figure 5: Illustration of the merging process of a linear layer with batch normalization.

This method extracts the signature of a targeted neuron using the critical points obtained from Algorithm 1, under the assumption that a full-rank system can be constructed. This implies that the number of critical points, *i.e.*, the number of equations, must be at least equal to the number of targeted parameters (weights and bias), and that each targeted parameter must appear with a non-zero coefficient in at least one equation. Unfortunately, there exist neurons for which these conditions are difficult to satisfy, *i.e.* neurons requiring a very large number of queries to identify a suitable subset of critical points. Such neurons can be broadly classified into two categories:

- The first category corresponds to neurons that almost never change state, except on a very small subset of the input space $\mathcal{X}$. Finding critical points through random sampling is therefore particularly challenging. Examples include *always-off* and *always-on* neurons, as introduced in Subsection 3.2.
- The second category, which has not been previously reported in the literature, corresponds to neurons for which one of the input entries is almost always null. For such neurons, the number of queries required to construct a full-rank matrix can exceed that of standard neurons by a factor of up to 10 (see Subsection 6.2). We refer to these neurons as *input-off*.

The presence of these neurons is correlated with the type of activation function used. In particular, the ReLU activation function is known to induce *always-off* neurons [Lu20], and we hypothesize that the repeated application of this activation function is a key factor contributing to the emergence of *input-off* neurons in DNNs. For these special-case neurons, the signature extraction process must be adapted. Once a special-case neuron is detected based on the unusually large number of queries required during extraction, a specific procedure must be applied. Since these neurons arise in ReLU-based DNNs, we now describe the extraction procedures tailored to each case.

Extraction of *always-off* neurons. The simplest special case is that of an *always-off* neuron. Since its output is almost always null and therefore independent of the input, its output can be directly set to 0, without extracting the associated weights.

Extraction of *always-on* neurons. For *always-on* neurons, the search for critical points can be extremely long and query-intensive. Moreover, due to the sequential nature of our attack, applying our method to a layer without fully extracting the preceding layers is not possible. It is therefore essential to extract all neurons at each layer. Knowing that a neuron is *always-on*, one approach is to treat its activation function, *i.e.* the ReLU, as the identity function and model the neuron as a weighted skip connection between the current

layer and the next. We illustrate this approximation for an *always-on* neuron η in the first extracted layer, as part of the signature associated with neuron λ in the subsequent layer, in Figure 6.

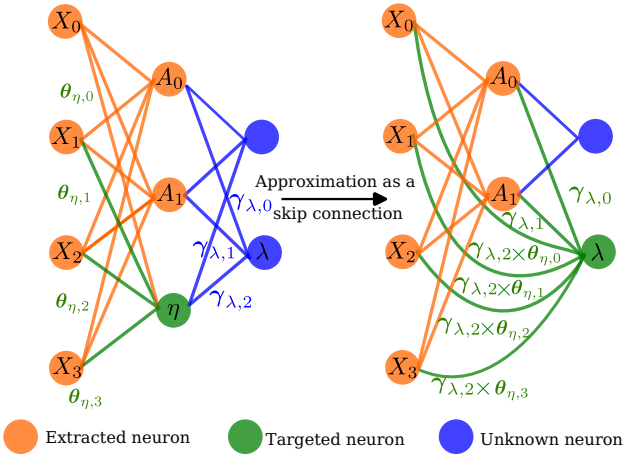


Figure 6: Approximation of the special-case neuron η as a skip connection to the neuron λ.

Let $\boldsymbol{X} \in \mathbb{R}^{n+1}$ be the input to a layer ℓ composed of m neurons, and let η be an *always-on* neuron in this layer with weight vector $\boldsymbol{\theta}_\eta \in \mathbb{R}^{n+1}$ and bias $\beta_\eta \in \mathbb{R}$. Let λ denote a neuron in layer $\ell+1$ with weight vector $\boldsymbol{\gamma}_\lambda \in \mathbb{R}^{m+1}$ and bias $\beta_\lambda \in \mathbb{R}$. If we denote by $\boldsymbol{A}_i$ the activation of neuron i in layer ℓ, neuron η can be approximated as a skip connection to λ, and Equation 1 becomes

$$\sum_{\substack{i=0,\\ i\neq\eta}}^{m} \gamma_{\lambda,i}\boldsymbol{A}_i + \gamma_{\lambda,\eta} \times \sum_{j=0}^{n} \boldsymbol{\theta}_{\eta,j}\boldsymbol{X}_j + \gamma_{\lambda,\eta}\beta_\eta + \beta_\lambda = 0. \tag{2}$$

For the example shown in Figure 6, Equation 2 becomes

$$\sum_{i=0}^{1} \gamma_{\lambda,i}\boldsymbol{A}_i + \gamma_{\lambda,2} \times \sum_{j=0}^{3} \boldsymbol{\theta}_{\eta,j}\boldsymbol{X}_j + \gamma_{\lambda,2}\beta_\eta + \beta_\lambda = 0, \boldsymbol{X} \in \mathbb{R}^4. \tag{3}$$

As shown above, Equation 2 remains linear with respect to the targeted neuron's weights, allowing the construction of a new linear system. Out of simplicity, we consider that η corresponds to the first neuron, *i.e.*, at index 0, then, using the same notation as before, this system can be expressed as follows:

$$\begin{cases} -\sum_{i=1}^{m} \frac{\gamma_{\lambda,i}}{\gamma_{\lambda,0}\boldsymbol{\theta}_{\eta,0}}\boldsymbol{A}_i^1 - \sum_{j=1}^{n} \frac{-\boldsymbol{\theta}_{\eta,j}}{\boldsymbol{\theta}_{\eta,0}}\boldsymbol{X}_j^1 - \frac{\beta_{\eta\lambda}}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^1 \\ \vdots \\ -\sum_{i=1}^{m} \frac{\gamma_{\lambda,i}}{\gamma_{\lambda,0}\boldsymbol{\theta}_{\eta,0}}\boldsymbol{A}_i^z - \sum_{j=1}^{n} \frac{-\boldsymbol{\theta}_{\eta,j}}{\boldsymbol{\theta}_{\eta,0}}\boldsymbol{X}_j^z - \frac{\beta_{\eta\lambda}}{\boldsymbol{\theta}_{\eta,0}} = \boldsymbol{X}_0^z \end{cases} \tag{4}$$

where $\beta_{\eta\lambda}$ denotes an aggregate of the biases of neurons η and λ, and z is the total number of equations, with $z \geq n+m+1$.

As this system closely resembles that of Equation 1, we solve it using the same approach. Notably, the signature of neuron η appears implicitly in the extracted signature of neuron

λ. Thus, by solving the LSTSQ problem for a neuron in the subsequent layer, it is possible to recover the signature of an *always-on* neuron. The only exception is the bias, which cannot be directly separated. Instead, the solution yields a combination of the biases of η and λ. Since the weight signatures are already known, only the bias remains unknown. Consequently, a single additional non-null critical point associated with the targeted *always-on* neuron suffices to extract its bias.

Extraction of *input-off* neurons. Finally, the extraction of *input-off* neurons follows a strategy similar to that used for *always-on* neurons. Here, the signature of a deeper neuron is leveraged to recover the signature of the targeted neuron. However, unlike *always-on* neurons, *input-off* neurons exhibit state changes depending on the input, preventing the activation function from being approximated as the identity for all inputs. Since we target ReLU activations, the activation function is either null or the identity. We therefore rely on the oracle to infer the neuron's state and determine which inputs correspond to each case, allowing us to manually adjust the input equations accordingly. We then solve the same system of equations as for *always-on* neurons to extract the neuron's signature. As in the *always-on* case, the bias is recovered using a critical point directly associated with the targeted *input-off* neuron.

Using the side-channel oracle, it is therefore possible to extract the signature of a given neuron while increasing robustness to special-case neurons. However, the signature extraction process assumes that the neuron's state can be reliably inferred and that the specific type of special-case neuron is correctly identified. The first assumption is experimentally validated in Subsection 5.2, while the neuron type identification is discussed in Subsection 4.3. To evaluate the impact of special-case neurons on the model extraction process, an additional analysis is conducted in Subsection 6.2, comparing the number of queries required to extract standard and special-case neurons. Assuming that the signature extraction is correctly performed, a final step remains: estimating the sign of the scaling factor in order to fully extract the ℓ-th layer.

4.3 Estimation of the sign of the scaling factor

The extraction of the sign using the side-channel oracle O follows a two-step hypothesis-testing process applied simultaneously to all neurons in the ℓ-th layer. First, the neurons in the layer are aligned to reduce the number of hypotheses to a single one for layer ℓ; second, the correct hypothesis is selected using information from the $(\ell+1)$-th layer. Furthermore, as discussed in Subsection 4.2, special-case neurons (e.g., Figure 6) effectively shift the extraction of a neuron to the $(\ell+1)$-th layer, and therefore require extracting the sign of special-case neurons. The method proposed in this section is designed to handle these constraints.

Layer alignment. The extraction of the sign of all scaling factors begins once all neuron signatures in the ℓ-th layer have been extracted. At this stage, the attacker knows the induced hyperplane for each neuron and can arbitrarily assign a state to each side of the hyperplane, denoted V_A and V_B, resulting in 2^n possible hypotheses, where n is the number of neurons in the layer. To reduce this number, the neurons are aligned such that, for each neuron in the layer ℓ, the arbitrary state V_A corresponds to the same real neuron state (either active or inactive), while V_B corresponds to the opposite state. This reduces the number of hypotheses to a single candidate corresponding to the identification of the sign associated with V_A or V_B. This alignment is performed using the side-channel oracle O. Indeed, by comparing the states of two neurons η and γ for a given input $\boldsymbol{X}$

(*i.e.* $V(\eta; \boldsymbol{X})$ and $V(\gamma; \boldsymbol{X})$), it is possible to align the scaling factors associated with the signatures of these two neurons.

Hypothesis selection. Once the neurons of the ℓ-th layer have been aligned, an arbitrary assumption is made regarding the states V_A and V_B; for instance, V_A (resp. V_B) is assumed to correspond to the active (resp. inactive) state. The validity of this hypothesis is then evaluated during the signature extraction of the first neuron in the $(\ell+1)$-th layer. This approach is justified as follows: once a sufficient number of critical points has been extracted in the $(\ell+1)$-th layer, a linear relationship is expected between these points and the output of the ℓ-th layer (see Subsection 4.2). An incorrect assumption on V_A and V_B breaks this linear relationship, as the ReLU function computes $\boldsymbol{A} = \texttt{ReLU}(-\boldsymbol{X})$ instead of $\boldsymbol{A} = \texttt{ReLU}(\boldsymbol{X})$. This results in incorrect activation values, *i.e.* incorrect inputs to the $(\ell+1)$-th layer, such that Equation 1 no longer holds. Consequently, the residual error of the LSTSQ solution (see Subsection 4.2) is used as a criterion to validate the sign hypothesis. The correct hypothesis is the one that yields the smallest residual error.

Using this method, we can extract the sign of each neuron without requiring additional queries. Furthermore, since the method requires only one hypothesis per layer, it remains practical even for complex and deep architectures. The final step that must be justified is the ability to retrieve the sign of special-case neurons.

Identifying the type of special-case neuron. By definition, special-case neurons cannot appear in the first layer of the targeted DNN, since the attacker controls the inputs sent to the network. Consequently, when a special-case neuron η is detected in layer ℓ, the preceding layer $(\ell-1)$ has already been fully extracted. Because this previous layer is known, the sign of its scaling factors is also known. The attacker can therefore select any neuron γ in layer $(\ell-1)$ and use the side-channel oracle O to compare the states of neurons η and γ for a given input. Depending on the oracle's output, the attacker can directly determine the state of the special-case neuron and, as a result, identify its type.

With this oracle, it is therefore possible to extract the critical points, the signature, and the sign of the scaling factor while allowing the extraction of special-base neurons. In the next section, we demonstrate how this divide-and-conquer methodology can be used to extract pooling layers for the first time.

4.4 Discussion on the methodology

4.4.1 Pooling layer extraction

Under the assumption that an attacker can extract the signatures and the signs of the scaling factors using the divide-and-conquer methodology, it becomes possible to target modern architectures composed of non–fully connected layers, such as MobileNets [HZC+17]. In Figure 7, we provide a high-level description of the architecture of *MobileNetv1*. This model consists of a sequence of depthwise separable convolutions, followed by an average pooling layer, and finally a classification block containing a fully connected layer and a softmax function. Due to the presence of the average pooling layer, such architectures have not previously been targeted by cryptanalytic-based methods [JCB+19, RK19, CJM20], as discussed in Subsection 3.2.

Based on the methodology introduced in Subsection 3.3, an attacker can leverage the side-channel oracle to target convolutional operations. As with fully connected layers, this process consists of targeting the state of each neuron induced in the depthwise and pointwise convolutional layers. Since the convolutional layers can be represented as a

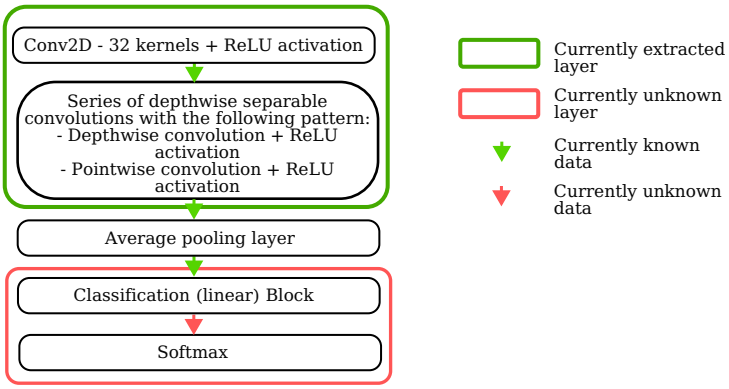


Figure 7: Extraction process of an architecture including an average pooling layer. (**Note**: It is assumed that all depthwise separable convolutions have been successfully extracted using our methodology).

special kind of fully connected layer with a very particular sparsity and weight-sharing pattern, the divide-and-conquer methodology can be straightforwardly applied in this setting. Additionally, as activation functions are used in such layers, the neuron's state can be retrieved equivalently. Moreover, in MobileNetv1, the pooling layer is placed immediately after an activation function, which allows its impact to be ignored when using the oracle. To illustrate this point, consider the case where all preceding layers have been successfully extracted. As shown in Figure 7, the input to the average pooling layer is then known; because this layer is non-parametric, its output is also known, and the attacker can directly target the subsequent layer. Consequently, pooling layers can be disregarded in such configurations, as they correspond to known transformations applied to known data. This demonstrates that complex architectures, including those with pooling layers such as MobileNetv1, can be effectively targeted. While we have described the benefits of using the side-channel oracle to target more complex architectures, the following section discusses the choice of oracle and argues that the divide-and-conquer strategy could benefit from other types of oracles.

4.4.2 Choice of the oracle

Our methodology relies on an oracle O that returns a boolean value indicating whether two neurons are associated with the same state for two given inputs. Any function achieving this goal can be used as an oracle within our framework. As a result, the methodology is not restricted to side-channel attacks or to the ReLU activation function. For instance, one could design an oracle based on fault-injection techniques by testing the activity of a neuron through the absence or presence of variations in the DNN output after faulting the pre-activation value, in a manner similar to the approach proposed in [HDMD23] for weight extraction. Similarly, if an attacker targeting other activation functions, such as tanh or ReLU6, is able to construct a function that distinguishes between different activation states, then our methodology remains applicable. It is worth noting that the activation states do not necessarily need to be binary or localized at 0, as is the case for the ReLU function. For example, in the case of ReLU6, if the attacker can distinguish between the inactive state (*i.e.* negative pre-activation values), the identity state (*i.e.* activation values between 0 and 6), and the constant state (*i.e.* activation values greater than 6), the methodology can be adapted by incorporating additional equations derived from points located between the identity and constant states. In such a scenario, the threat model would no longer be restricted to DNNs using the ReLU activation function. We leave the

exploration of alternative oracles and activation functions for future work.

5 Practical experiments

In this section, we test our methodology by first validating our side-channel-based oracle and then performing the extraction of DNNs. We begin with a description of the targeted architectures in Subsection 5.1. Then, in Subsection 5.2, we describe the vulnerability in the ReLU function and evaluate the effectiveness of our side-channel-based oracle. Our results on classifiers are presented in Subsection 5.3, and we provide a comparison between our methodology and soft-label extraction methods in Subsection 5.4. Finally, in Section 6, we discuss the perspectives and limitations of our results.

5.1 Experimental setup

Dataset and Architecture. We target several embedded DNNs, including a shortened MobileNetv1 [HZC+17]. A detailed description of this model is provided in Table 9. This DNN shares the same basic blocks as the original MobileNetv1, namely depthwise separable convolutions. The architecture of a depthwise separable convolution is described in Table 10. To the best of our knowledge, these blocks have never been successfully extracted using cryptanalytic extraction methods. We also tested our methodology on a large MLP with the architecture described in Table 11.

For clarity, when describing MLPs, we adopt the following notation: Size Input - (Size Hidden Layer) - Size Output Layer. For instance, the large MLP described above is denoted as 3072-512-256-64-10. Finally, we target the most complex architectures considered in [JCB+19, RK19, CJM20, CDG+25, CCSH+25]. A summary of the targeted architectures is provided in Table 2.

Table 2: Summary of the targeted architectures.

Type of architecture	Nb. of parameters	Nb. of targets	Targeted by
Small MLP	$< 200,000$	7	[CJM20, JCB+19, RK19, CDG+25], This work
MLP	$935,370$	1	[CCSH+25], This work
Large MLP	$1,721,802$	1	This work
MobileNetv1-short	$5,234$	1	This work

All classifiers are trained on the CIFAR-10 dataset[3] except for the architecture from [CDG+25]. This network, as well as all DNNs performing regression, is trained on a random dataset, as in the SOTA.

Hardware configuration. The models are deployed on an STM32-F767ZI board, based on an ARM Cortex-M7 MCU, via the X-Cube-AI framework without compiler optimizations, using 32-bit floating-point data for all values, *e.g.* inputs, weights, and activation values. The board features 2 MBytes of Flash memory and 512 KBytes of SRAM. X-Cube-AI is a framework developed by ST Microelectronics to facilitate DNN deployment on STM's embedded devices. However, this framework is not designed to provide secure deployment of embedded DNNs.

Practical setup. An EM probe from Langer (EMV-Technik RF-U 2.5) is connected to an amplifier (ZLF-2000G+) to capture EM traces. A Lecroy oscilloscope (2.5 GHz

[3] https://www.cs.toronto.edu/~kriz/cifar.html

WaveRunner 625Zi) is used to acquire the EM signal. To simplify leakage exploitation, a trigger surrounding the ReLU processing is added. Since the patterns around the ReLU function are distinguishable from those of the preceding matrix product, the trigger configuration is not considered difficult and does not alter the threat model considered in this paper.

5.2 Side-channel analysis of the ReLU implementation

Our methodology relies entirely on the oracle's ability to distinguish the state of two inputs for a given neuron. Thus, we first validate that such a function can be constructed using the distinguisher introduced in Subsection 3.3. Here, we present results obtained using the setup described above.

Description of the vulnerability. The ReLU implementation in X-Cube-AI is constant-time, as recommended in [MBC21]. Algorithm 2 illustrates the main idea for 32-bit values[4]. For a signed pre-activation value $z \in \mathbb{R}$ encoded on 32 bits, the sign can be

Algorithm 2 Pseudo-code of the constant-time ReLU function.

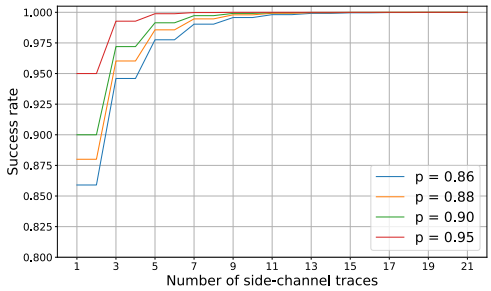
Require: Pre-activation value $z \in \mathbb{R}$
Ensure: $a = \texttt{max}(0, z) \in \mathbb{R}^+$
$a \leftarrow (\sim (z \gg 31))$ & z

extracted via $(z \gg 31)$ such that it is then cast to 32 bits to serve as a mask, *i.e.* $(\sim (z \gg 31)) \in \{\texttt{0x00000000, 0xFFFFFFFF}\}$. Depending on the sign of z, applying this mask either leaves the value unchanged or sets the output to zero. Although the implementation is constant-time, the mask operation produces a physical leakage exploitable by an attacker. Specifically, the mask is either `0x00000000` or `0xFFFFFFFF` depending on the neuron's state. The difference in Hamming weights between these two values allows separation into two groups using clustering algorithms, corresponding to the active and inactive states.

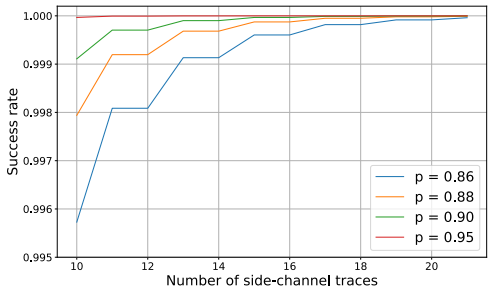
Critical point search. To validate our method, we acquire $10,000$ EM traces using the setup in Subsection 5.1. We then follow the strategy in Subsection 3.3 and measure the success rate of retrieving the state of 12 neurons. The *k-means* algorithm is used to cluster the neuron states without prior knowledge of their true states. As noted in Subsection 3.3, only correct separation between the two clusters is required for the attack. We compute the success rate by comparing the cluster assignments to the true neuron states, *i.e.* verifying that neurons with the same state are assigned to the same cluster. Using one EM trace, we achieve an average success rate of 85.9%. While this validates our methodology, high query complexity (as in [JCB+19], see Subsection 4.1) motivates achieving near-100% success for complex DNNs. A simple way to improve success is to acquire multiple traces for the same input and use majority voting to infer the state.

Figure 8a shows the success rate obtained using majority voting versus the number of traces for different initial success probabilities. With an initial success of 85.9%, the success rate exceeds 99% with 7 traces and 99.9% with 13 traces. For worst-case analysis, we assume an ideal scenario where one trace suffices for a 100% success rate. All results in the following sections are reported under this assumption, providing an empirical upper bound for performance. This assumption is not restrictive, since if N traces are required to achieve 100% success, the overall complexity scales linearly with N. Finally, as these

[4]Understanding the vulnerability is not required to perform our black-box extraction attack; we present it here for illustration purposes.



(a) Success rate with different initial probabilities.



(b) Zoom of Figure 8a after ten traces in the same settings.

Figure 8: Evolution of the success rate of neuron state extraction as a function of the number of side-channel traces.

results depend on the targeted device, *i.e.* other devices may produce lower success rates, we discuss the impact of oracle error rates in Subsection 6.4.

Sign extraction. We also validate that the oracle can align neuron states for sign extraction. Using the method in Subsection 4.3, the layer-level clusters are reduced to two. This achieves a success rate of 99.7% for retrieving the sign of each neuron's scaling factor from a single trace. Our results are independent of the clustering algorithm and could be improved using other unsupervised methods. Since the vulnerability originates from the software implementation, any platform using it is a potential target. To our knowledge, this represents the first successful extraction of neuron scaling factor signs in a hard-label setting without additional queries.

5.3 Complete extraction of classifiers

Evaluation metrics. To evaluate the fidelity of the extracted models, we compute the metrics introduced in [RCYF21] and [HDMD23]: *Fidelity* and *Accuracy Under Attack* (AuA). Fidelity metric measures the average label agreement between the copied model and the original model on the *top-1* prediction for a subset of inputs. AuA measures the transfer rate of adversarial examples generated on the copied model and tested on the targeted model, serving as a proxy for the ability to craft effective attacks on the victim's network using the copied model. We also compute the maximum absolute difference between the true weights θ and their extracted counterparts $\hat{\theta}$ to characterize extraction success, as in [JCB+19, RK19, CJM20]. This metric is computed both on the entire network of depth L ($\texttt{max}|\Delta_\theta|^L$) and on the first $L-1$ layers ($\texttt{max}|\Delta_\theta|^{L-1}$). Finally, the number of queries required to the targeted model is used to evaluate the cost of the attack.

Summary of the complete results. To our knowledge, cryptanalytic extraction of DNNs in hard-label settings has previously only been performed in [CDG+25, CCSH+25]. We first test our methodology on the largest architectures from these works: a 1024-2-2-1 MLP [CDG+25] and a 3072-256-256-256-64-10 MLP [CCSH+25], referred to as MLP-a and MLP-b, respectively. It is important to note that while [CCSH+25] extracts the hidden layers, it does not extract the last layer, making it impossible to report metrics for the full network, *i.e.*, number of queries, fidelity, and $\texttt{max}|\Delta_\theta|^L$. We additionally extract a 3072-512-256-64-10 MLP (MLP-c) with 1.7 million parameters, nearly doubling the previous maximum number of parameters extracted in SOTA studies. The results are summarized in Table 3.

Table 3: Comparison in hard-label settings.

Architecture	Parameters	Approach	Queries	$\texttt{max}\lvert\Delta_\theta\rvert^{L-1}$	$\texttt{max}\lvert\Delta_\theta\rvert^{L}$	Fidelity
MLP-a 1024-2-2-1 (64-bit data)	**2,059**	[CDG+25]	$2^{22.49}$	-	$\mathbf{2^{-20.38}}$	**100%**
		This work	$\mathbf{2^{17.12}}$	$\mathbf{2^{-42.5}}$	$2^{3.45}$	99.5%
MLP-b 3072-256-256-256 -64-10 (64-bit data)	**935,370**	[CCSH+25]	-	$2^{-18.0}$	-	-
		This work	$\mathbf{2^{26.2}}$	$\mathbf{2^{-26.6}}$	$2^{2.1}$	97.2%
MLP-c 3072-512-256-64-10 (32-bit data)	**1,721,802**	**This work**	$2^{26.0}$	$2^{2.3}$	$2^{3.2}$	93.2%
Truncated MobileNetv1 (32-bit data)	**5,234**	**This work**	$2^{18.6}$	$2^{-4.3}$	$2^{3.7}$	88.4%

Performance comparison. We successfully extract the architectures mentioned above, achieving 99.5% fidelity for MLP-a and 97.2% for MLP-b. Compared to [CDG+25], our methodology reduces the number of queries by a factor of ≈ 41 while maintaining comparable fidelity (100% in [CDG+25] vs 99.5% in our work). This drop in the fidelity is due to the fact that their methodology is specifically designed for binary classifiers. Indeed, for binary classifiers, it is possible to form a system of equations using decision boundary points, *i.e.*, points on the boundary where the output label changes, and the activation value of the previous layer. While this allows for precise extraction of the last layer, it is not directly applicable to all multiclass classifiers. Therefore, we use our method, which is more generic.

Since Carlini *et al.* [CCSH+25] do not perform complete extraction of MLP-b, we can only evaluate the maximum error on the hidden layer weights. Using this metric, our method achieves a more accurate estimation by a factor of 388. In addition, [CCSH+25] assumes perfect extraction of the previous layer when targeting a new one, removing residual error propagation. For comparison, we simulated our methodology under the same assumption and computed the maximum absolute difference with and without residual error, as shown in Table 4.

Table 4: Maximum error on the weights with and without the residual error on the MLP architecture from [CCSH+25]

Assumption	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
With residual error	$2^{-38.1}$	$2^{-33.8}$	$2^{-29.7}$	$2^{-26.6}$	$2^{2.1}$
Without residual error	$2^{-38.1}$	$2^{-41.9}$	$2^{-42.8}$	$2^{-41.3}$	$2^{2.0}$

Under the assumption of no error propagation, performance is consistent across layers, and our method further improves the results of [CCSH+25], reducing the maximum hidden layer weight error by a factor of $\approx 2^{20}$. Finally, even when considering error propagation, we achieve high-fidelity extraction of the new architectures: 88.4% for MobileNetv1 and 93.2% for MLP-c on the full CIFAR-10 test set ($10,000$ images).

Comparison with *learning-based* extraction. We compare our results with *learning-based* extraction methods using a baseline model trained on a balanced dataset composed of the queries used during the critical points' search, with the responses of the targeted model as labels. The baseline model uses the same hyperparameters as the targeted model, *e.g.*, architecture and learning rate. While the baseline approach achieves reasonable performance during training (around 56% accuracy), it generalizes poorly, and its performance drops significantly on the test set, achieving only 21.1% fidelity for MobileNetv1. The use of random queries, which do not follow the same distribution as the true CIFAR-10 training data, combined with the hard-label setting, drastically reduces the baseline model's performance. This illustrates that *learning-based* extraction methods are heavily dependent on data distribution, highlighting the advantage of cryptanalytic attacks for achieving higher fidelity in the threat model considered in this work.

Generation of adversarial examples. To complete our performance evaluation, we use AuA to quantify how much our extracted model can improve other attacks, particularly adversarial attacks. We generate adversarial examples using the copied model and measure their impact on the targeted model, employing three different attack algorithms. We first use the PGD attack [MMS+18] with the settings from [RCYF21, HDMD23], *i.e.* a perturbation factor of 8/255 and 40 iterations. We also generate adversarial examples using the BIM attack [KGB18] with up to 4 iterations and a gradient step of 5×10^{-3}. Finally, we use the C&W attack [CW17] with the L2 norm, using the CleverHans implementation [PFC+18], a learning rate of 5×10^{-3}, and a maximum of 1000 iterations. All the results are reported in Table 5. For MLP-b, MLP-c and MobileNetv1, the transfer rates are close to white-box performance across all evaluated methods. These results outperform a powerful substitute model trained with access to the true data [RCYF21], demonstrating that our methodology enables more effective attacks in hard-label settings.

Table 5: Transfer rates (in term of AuA) of adversarial examples generated on the copied model and tested on the targeted model.

Architecture	**PGD** [MMS+18]	**BIM** [KGB18]	**C&W** [CW17]
MLP-b 3072-256-256-256-64-10 (64-bit data)	98.04%	98.71%	89.61%
MLP-c 3072-512-256-64-10 (32-bit data)	97.48%	98.40%	90.27%
Truncated MobileNetv1 (32-bit data)	98.62%	97.30%	98.01%

5.4 Simulation of our methodology in soft-label setting

To ensure a fair comparison between our methodology and other cryptanalysis-based approaches, we include results from SOTA on DNNs trained for regression tasks, assuming access to the continuous output value. In this context, we leverage this access to improve the extraction of the last layer, which differs from the previous layers, as mentioned in Subsection 3.3. In contrast, previous methodologies use this output to extract the hidden layers directly and can perform gradient-based optimizations to improve extraction efficiency. However, none of these works target DNNs in hard-label settings, unlike our methodology.

Evaluation metrics. For regression tasks, metrics such as AuA and fidelity, *i.e.*, label agreement, are no longer meaningful. Consequently, SOTA methodologies evaluate extraction fidelity by considering only the maximum error on the extracted weights [JCB+19, RK19, CJM20]. This metric has the advantage of providing an estimate of fidelity that is independent of any assumptions about the input distribution, unlike a layer-wise mean squared error. Finally, we continue to use the number of queries to quantify the cost of the attack.

As our methodology is not based on the gradient of the targeted DNN, it is not affected by the precision of the output format. Previous methodologies target DNNs using 64-bit data. In this section, we first present results obtained from simulations using 64 bits and then provide results using 32-bit data. We use the same notation for describing the architecture as introduced in Subsection 5.1, *i.e.*, a 784-128-1 DNN corresponds to a model with an input size of 784, one hidden layer composed of 128 neurons, and one output.

Results on DNN using 64-bit data. Table 6 provides a comparison of DNNs trained for regression in soft-label settings with 64-bit data. In terms of extraction fidelity, our methodology outperforms all previous methods, reducing the maximum error on the

weights by factors ranging from $2^{4.5}$ to $2^{16.1}$, despite operating in hard-label settings for the hidden layers. The side-channel attack allows for more precise localization of critical points compared to gradient-based methods, providing a more accurate estimation of the weights. Regarding attack cost, our methodology requires fewer queries for four out of the six previously targeted architectures. For the two architectures where our attack is more costly, we hypothesize that this is due to their narrow topology, *i.e.* only one hidden layer, which benefits gradient-based optimizations used in SOTA methodologies. For the remaining architectures, the direct mapping from critical points to neurons provided by the side-channel attack improves our methodology's efficiency relative to SOTA.

Table 6: Comparison of DNNs trained for regression in soft-label settings with 64-bit data.

Architecture	Parameters	Approach	Queries	$\max\|\boldsymbol{\Delta}_\theta\|^L$
10-10-10-1	210	[CJM20]	$2^{16.0}$	$2^{-36.0}$
		[RK19]	$2^{22.0}$	$2^{-12.0}$
		This work	$\mathbf{2^{15.6}}$	$\mathbf{2^{-46.2}}$
10-20-20-1	620	[CJM20]	$2^{17.1}$	$2^{-37.0}$
		This work	$\mathbf{2^{15.6}}$	$\mathbf{2^{-46.5}}$
40-20-10-10-1	1,110	[CJM20]	$2^{17.8}$	$2^{-27.1}$
		This work	$\mathbf{2^{16.8}}$	$\mathbf{2^{-42.0}}$
80-40-20-1	4,020	[CJM20]	$2^{18.5}$	$2^{-39.7}$
		This work	$\mathbf{2^{18.3}}$	$\mathbf{2^{-44.2}}$
784-32-1	25,120	[CJM20]	$2^{19.2}$	$2^{-30.2}$
		[JCB+19]	$\mathbf{2^{18.2}}$	$2^{-1.7}$
		This work	$2^{20.6}$	$\mathbf{2^{-43.5}}$
784-128-1	100,480	[CJM20]	$\mathbf{2^{21.5}}$	$2^{-24.7}$
		This work	$2^{22.6}$	$\mathbf{2^{-40.8}}$

Results on DNN using 32-bit data. Table 7 provides a comparison of our methodology against targeted architectures from SOTA methodologies using 32-bit data. Although no prior comparisons exist in this setting, we present these results for future reference. We successfully extract all architectures under these conditions as well.

Table 7: Results of our methodology against targeted architectures by SOTA methodologies using 32-bit data.

Architecture	Parameters	Queries	$\max\|\Delta_\theta\|^L$
10-10-10-1	210	$2^{13.0}$	$2^{-18.2}$
10-20-20-1	620	$2^{14.5}$	$2^{-17.8}$
40-20-10-10-1	1,110	$2^{16.4}$	$2^{-12.1}$
80-40-20-1	4,020	$2^{19.1}$	$2^{-14.8}$
784-32-1	25,120	$2^{19.8}$	$2^{-17.7}$
784-128-1	100,480	$2^{21.7}$	$2^{-17.4}$

While our methodology achieves strong results, Table 6 and Table 7 indicate that both the depth and the number of neurons per layer affect the attack's performance. In the next section, we further analyze this limitation and evaluate the impact of special-case neurons, as well as the extraction of the last layer.

6 Discussion of experimental part

6.1 Propagation of errors

In Subsection 3.3, we introduced our strategy as a divide-and-conquer approach, which requires knowledge of the $(i-1)$-th layer to extract the i-th one. Thus, one may question

Table 8: Layer-wise statistics of the extraction of the shortened MobileNetv1.

Datatype	Metrics	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
32 bits	$\texttt{max}\lvert\theta - \hat{\theta}\rvert^L$	$2^{-18.9}$	$2^{-17.6}$	$2^{-7.9}$	$2^{-18.2}$	$2^{-7.6}$	$2^{-13.9}$	$2^{-9.9}$	$2^{-11.4}$	$2^{-8.8}$	$2^{-6.7}$	$2^{-4.3}$	$2^{3.7}$
64 bits	$\texttt{max}\lvert\theta - \hat{\theta}\rvert^L$	$2^{-46.6}$	$2^{-43.8}$	$2^{-37.4}$	$2^{-34.2}$	$2^{-29.0}$	$2^{-27.1}$	$2^{-26.0}$	$2^{-26.8}$	$2^{-23.1}$	$2^{-22.7}$	$2^{-15.3}$	$2^{3.8}$

the impact of error propagation if the $(i-1)$-th layer is not successfully extracted. As mentioned in Subsection 4.1, using binary search, it is only possible to place a critical point within a certain distance of the hyperplane. This introduces a small error in the estimation of the weights, as illustrated in Table 8, which provides the layer-wise statistics of the shortened MobileNetv1 extraction. The propagation of this small error in the current layer's weights accumulates in the activation values, *i.e.*, the inputs for the system of equations of the next layer, which reduces the extraction performance of subsequent layers. This pattern is clearly visible in Table 8. For the first layer L0, the maximum error on the estimated weights is $2^{-18.9}$, while for the last hidden layer, this error increases to $2^{-4.3}$. For comparison, we also show results using 64-bit data. We observe the same pattern; the difference is that higher precision in localizing critical points reduces the error after the first layer to $2^{-46.6}$, compared to 32-bit data, resulting in a lower final error of $2^{-15.3}$. This error propagation is not intrinsic to our method, as all SOTA cryptanalytic extraction methodologies are affected by it. It is therefore crucial to minimize the error at each layer to reduce its impact on subsequent layers, highlighting the benefits of using an oracle-based methodology. Alternatively, several methods could be implemented to reduce this propagation, such as re-computing the weights to achieve higher precision. However, we leave this improvement for future work.

6.2 Impact of the special-case neurons

Another point of interest is the impact of special-case neurons. Through the oracle, it is possible to improve the robustness of the extraction process, but their extraction remains costly in terms of queries. To visualize their impact, we present the share of special-case neurons in the total number of requests and neurons for the 3072-256-256-256-64-10 MLP in Figure 9. While previous works mention the existence of *always-off* and *always-on* neurons, their presence is not considered during extraction, and existing methodologies do not propose solutions for these neurons. This is especially true for *input-off* neurons, which are not even acknowledged in SOTA. However, we observe such neurons in our experiments, even in simple architectures. In particular, we compute the proportion of queries dedicated to special-case neurons per layer, as well as the proportion of these neurons per layer, as shown in Figure 9 for the 3072-512-256-64-10 MLP. While most of these special neurons correspond to *input-off* and could theoretically be extracted by previous methodologies, doing so greatly increases the cost of the attack compared to methods using an oracle to optimize extraction.

These results highlight the impact of special-case neurons on the number of requests required to extract a DNN, even for a simple architecture such as a five-layer MLP.

6.3 Extraction of the last layer

As previously mentioned, extraction differs for the last layer. This is why we include both the maximum error on the first $L-1$ layers and on all L layers, *i.e.*, the full network. Since there is no non-linear activation function for this layer, no critical points can be found. In [CDG+25], the authors leverage the fact that they target binary classifiers to precisely extract the last layer. To improve the generality of our methodology, we instead use supervised learning on a dataset composed of the random inputs generated for the previous

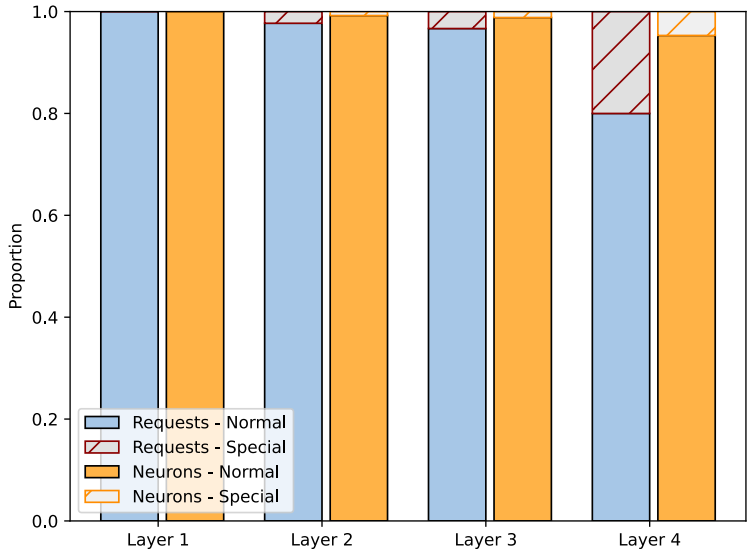


Figure 9: Comparison of the proportion of requests dedicated to normal versus special-case neurons.

layers and their corresponding hard-labels to train the last layer to behave similarly to the targeted model's last layer. While this methodology allows extraction of the last layer in hard-label settings, it accounts for most of the error in our process.

To validate this, we combine the first eleven extracted layers with the true final layer of the targeted model. This hybrid model achieves a near-perfect fidelity of 99.56%, confirming that the main loss in fidelity comes from the last layer extraction. As this limitation affects all methods based on extracting critical points in hard-label settings on multi-class classifiers, improving last-layer extraction is left for future work. Two alternative methods could mitigate this issue. The first is the methodology in [CCSZ23], performing a side-channel attack to retrieve confidence scores before extracting the last layer, which requires a profiling device and thus changes the threat model. The second is the methodology in [CMS25], *i.e.*, fixing several parameters and extracting the last layer under these constraints. This approach could be used as a supplementary attack to improve final-layer fidelity. However, while it achieves high label agreement, the last layer is not completely extracted, and its impact on downstream attacks, such as evasion attacks, has not been investigated [CMS25].

6.4 Impact the error rate of the oracle

In Subsection 5.3 and Subsection 5.4, we assume access to a perfect oracle, enabling the attacker to perfectly compare the states of two neurons with a single EM trace. However, as shown in Subsection 5.2, this is not the case in practice, and the oracle may fail to compare neuron states. A single wrong estimation during binary search can produce an incorrectly estimated critical point, resulting in a biased system of equations that may deteriorate weight estimation or render the system unsolvable. On average, placing one critical point in our setup requires 40 oracle calls. If the oracle has an 85.9% success rate, only 0.2% of the estimated critical points are truly critical. With a 99.9% success rate, achievable in 13 EM traces, this percentage rises to 96.1%. This highlights the need for a near-perfect oracle with minimal side-channel traces. We identify two complementary strategies: reduce the oracle's error rate per trace and make the extraction process robust

to incorrect critical points. The first depends heavily on the targeted device; attackers could improve the oracle by physically accessing the chip to improve signal quality or enhance side-channel trace preprocessing. Although we use the k-means algorithm here, different clustering methods could further improve oracle accuracy. For the second strategy, our methodology could be adapted to consider equations from critical points as correct with probability p, reflecting oracle uncertainty. Under this adaptation, the attacker would sample extra critical points and solve the LSTSQ problem on subsamples, similarly to a RANSAC algorithm [FB81]. Validation of these adaptations is left for future work.

6.5 Countermeasures

Our methodology relies on the oracle O, *i.e.*, the side-channel attack on the ReLU function. If the attacker can no longer compare neuron states for specific inputs, our methodology becomes inapplicable. Therefore, countermeasures should specifically protect activation functions. While the constant-time implementation in [MBC21] exists, it remains vulnerable to other side-channel attacks, as demonstrated here. One option is to operate on masked values or use a LUT-based ReLU implementation. Another, without modifying the ReLU implementation, is to shuffle the neuron execution order at each run. Our binary search, see Subsection 4.1, only succeeds if the neuron execution order is known. Randomizing the order makes binary search and critical point collection more difficult to be performed. Evaluation of these countermeasures is left for future work.

7 Conclusion

In this study, we describe how side-channel attacks can be used as an oracle to improve fidelity-based model extraction attacks. We design a new methodology inspired by cryptanalysis, introducing a new paradigm: divide and conquer. With this approach, we demonstrate the extraction of new complex DNNs, including pooling layers, in hard-label settings while improving the performance of the extraction process. Finally, we validate our approach by simulating the oracle and performing the extraction of the most complex architecture targeted in soft-label settings, showing that, despite a stricter threat model, our approach remains competitive and even outperforms previous methodologies in terms of accuracy on the extracted weights.

Several potential directions for future work can be considered to improve our methodology. The first concerns the oracle. While we have used unsupervised clustering on the EM traces corresponding to the processing of the ReLU function on an MCU, our oracle methodology is neither restricted to this algorithm, *i.e.*, *k-means*, nor to this activation function, nor to this type of physical device. Therefore, our methodology could be extended to DNNs using any kind of activation function, provided it is possible to construct an oracle with the properties described in Subsection 4.1. Another challenge is to develop an oracle adapted to parallelized architectures such as GPUs, TPUs, or FPGAs. As previously mentioned, in these systems the processing is parallelized, which may make side-channel leakages more difficult to exploit. One possibility is to change the type of side-channel entirely by using time-to-digital converters, as in [LSC+25], or to consider alternative physical attacks and fault injection methods. The second concerns the adaptation of the methodology to other types of DNN architectures. While in our experiments we target a MobileNetv1 and several MLPs, a large variety of architectures remain unexplored and could be considered under the assumption that a suitable oracle has been identified.

Finally, the most important perspective, from our point of view, is to study the capacity of our methodology to target quantized DNNs. Indeed, in Subsection 6.1, we describe

how errors propagate from one layer to another and how the datatype used impacts the original error. To the best of our knowledge, no previous work has studied the impact of quantization on cryptanalytic extraction, making this a promising direction for future work.

Acknowledgement

Gabriel Zaid's research is funded by the European Union (ERC, AMAskZONE, 101077506). Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

A Description of the newly targeted architectures

An overall description of the targeted architecture based on MobileNetv1 is given in Table 9 and of the depthwise separable convolution in Table 10. Then in Table 11, we detail the architecture of our large MLP.

Table 9: Description of the architecture of the shortened MobileNetv1.

Layers	**Output size**	**Parameters**
Conv2D 3×3, padding, stride=2	$1 \times 12 \times 16 \times 16$	324
BatchNorm2D	$1 \times 12 \times 16 \times 16$	24
ReLU	$1 \times 12 \times 16 \times 16$	0
Depth-wise separable convolution	$1 \times 8 \times 16 \times 16$	244
Depth-wise separable convolution	$1 \times 16 \times 8 \times 8$	248
Depth-wise separable convolution	$1 \times 16 \times 8 \times 8$	464
Depth-wise separable convolution	$1 \times 32 \times 4 \times 4$	752
Depth-wise separable convolution	$1 \times 64 \times 4 \times 4$	2528
Adaptive Average Pooling	$1 \times 64 \times 1 \times 1$	0
Dense	1×10	650

Table 10: Description of a depth-wise separable convolution taking N_c channel as input.

Layers	**Kernel size**	**Padding**	**Grouped convolution**	**Bias**
Conv2D BatchNorm2D ReLu	3×3	1	✓	✗
Conv2D BatchNorm2D ReLu	$N_c \times 1 \times 1$	1	✗	✗

Table 11: Description of the architecture of the large MLP.

Layers	**Output size**	**Parameters**
Flatten	1×3072	0
Dense + ReLU	1×512	$1,573,376$
Dense + ReLU	1×256	$131,328$
Dense + ReLU	1×64	$16,448$
Dense + ReLU	1×10	650

References

[BBJP19] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel. In *USENIX Security Symposium*, 2019.

[CCP17] Eleonora Cagli, Cécile Canovas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *Workshop on Cryptographic Hardware and Embedded Systems*, 2017.

[CCSH+25] Nicholas Carlini, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, and Adi Shamir. Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 364–396, Cham, 2025. Springer Nature Switzerland.

[CCSZ23] Benoît Coqueret, Mathieu Carbone, Olivier Sentieys, and Gabriel Zaid. When side-channel attacks break the black-box property of embedded artificial intelligence. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023.

[CDG+25] Yi Chen, Xiaoyang Dong, Jian Guo, Yantian Shen, Anyu Wang, and Xiaoyun Wang. Hard-label cryptanalytic extraction of neural network models. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 207–236, Singapore, 2025. Springer Nature Singapore.

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In *Annual International Cryptology Conference*, 2020.

[CMCSH+24] Isaac A. Canales-Martínez, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, Nitin Satpute, and Adi Shamir. Polynomial time cryptanalytic extraction of neural network models. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 3–33, Cham, 2024. Springer Nature Switzerland.

[CMS25] Isaac A. Canales-Martínez and David Santos. Extracting some layers of deep neural networks in the hard-label setting. Cryptology ePrint Archive, Paper 2025/1118, 2025.

[CW16] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2016.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[DHG01] Richard Duda, Peter Hart, and David G.Stork. *Pattern Classification*. 01 2001.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[GQZ+24] Yansong Gao, Huming Qiu, Zhi Zhang, Binghui Wang, Hua Ma, Alsharif Abuadbba, Minhui Xue, Anmin Fu, and Surya Nepal. Deeptheft: Stealing dnn model architectures through power side channel. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3311–3326, 2024.

[HCW+25] Peter Horvath, Lukasz Chmielewski, Léo Weissbart, Lejla Batina, and Yuval Yarom. {BarraCUDA}: Edge {GPUs} do leak {DNN} weights. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 4017–4034, 2025.

[HDMD23] Kévin Hector, Mathieu Dumont, Pierre-Alain Moellic, and Jean-Max Dutertre. Fault injection and safe-error attack for extraction of embedded neural network models. In *ESORICS 2023 International Workshops*, volume 14399 of *Computer Security. ESORICS 2023 International Workshops CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25–29, 2023, Revised Selected Papers, Part II*, pages 644–664, La Hague, Netherlands, September 2023. Delft University of Technology.

[HR19] Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. In *Neural Information Processing Systems*, 2019.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[IMT25] Akira Ito, Takayuki Miura, and Yosuke Todo. Is the hard-label cryptanalytic model extraction really polynomial? Cryptology ePrint Archive, Paper 2025/1868, 2025.

[JCB+19] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alexey Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *USENIX Security Symposium*, 2019.

[JMPR22] Raphael Joud, Pierre-Alain Moëllic, Simon Pontié, and Jean-Baptiste Rigaud. A practical introduction to side-channel extraction of deep neural network parameters. In *Smart Card Research and Advanced Application Conference*, 2022.

[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 Shivam Bhasin and Thomas Roche, editors, *Smart Card Research and Advanced Applications - 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14-16, 2023, Revised Selected Papers*, volume 14530 of *Lecture Notes in Computer Science*, pages 256–276. Springer, 2023.

[KGB18] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[LSC+25] Guillaume Lomet, Rubén Salvador, Brice Colombier, Vincent Grosso, Olivier Sentieys, and Cédric Killian. Side-channel extraction of dataflow ai accelerator hardware parameters. In *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2025.

[LSE+25] Haolin Liu, Adrien Siproudhis, Samuel Experton, Peter Lorenz, Christina Boura, and Thomas Peyrin. Navigating the deep: Signature extraction on deep neural networks. *ArXiv*, abs/2506.17047, 2025.

[Lu20] Lu Lu. Dying ReLU and Initialization: Theory and Numerical Examples. *Communications in Computational Physics*, 28(5):1671–1706, June 2020.

[MBC21] Saurav Maji, Utsav Banerjee, and Anantha P. Chandrakasan. Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels—attacks and defenses. *IEEE Internet of Things Journal*, 8:12079–12092, 2021.

[MMS+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, pages 3–26. Springer, 2016.

[MSDH18] Smitha Milli, Ludwig Schmidt, Anca D. Dragan, and Moritz Hardt. Model reconstruction from model explanations. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2018.

[MXL+21] 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*, 2021.

[PFC+18] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[RCYF21] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1157–1174, 2021.

[RK19] David Rolnick and Konrad Paul Kording. Reverse-engineering deep relu networks. In *International Conference on Machine Learning*, 2019.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[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, 2020.

[ZDF23] Xiang Zhang, Aidong Adam Ding, and Yunsi Fei. Deep-learning model extraction through software-based power side-channel. *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.