Navigating the Deep: Signature Extraction on Deep Neural Networks

Haolin Liu[1,2], Adrien Siproudhis[2], Samuel Experton[3], Peter Lorenz[2], Christina Boura[3], and Thomas Peyrin[2]

Shanghai Jiao Tong University, China[1]
Nanyang Technological University, Singapore[2]
IRIF, Université Paris Cité, France[3]

arXiv:2506.17047v1 [cs.LG] 20 Jun 2025

Abstract. Neural network model extraction has emerged in recent years as an important security concern, as adversaries attempt to recover a network's parameters via black-box queries. A key step in this process is signature extraction, which aims to recover the absolute values of the network's weights layer by layer. Prior work, notably by Carlini et al. (2020), introduced a technique inspired by differential cryptanalysis to extract neural network parameters. However, their method suffers from several limitations that restrict its applicability to networks with a few layers only. Later works focused on improving sign extraction, but largely relied on the assumption that signature extraction itself was feasible.

In this work, we revisit and refine the signature extraction process by systematically identifying and addressing for the first time critical limitations of Carlini et al.'s signature extraction method. These limitations include rank deficiency and noise propagation from deeper layers. To overcome these challenges, we propose efficient algorithmic solutions for each of the identified issues, greatly improving the efficiency of signature extraction. Our approach permits the extraction of much deeper networks than was previously possible. We validate our method through extensive experiments on ReLU-based neural networks, demonstrating significant improvements in extraction depth and accuracy. For instance, our extracted network matches the target network on at least 95% of the input space for each of the eight layers of a neural network trained on the CIFAR-10 dataset, while previous works could barely extract the first three layers. Our results represent a crucial step toward practical attacks on larger and more complex neural network architectures.

Keywords: ReLU-based neural networks · signature extraction · weight-recovery

1 Introduction

Neural Networks (NNs) are a class of machine learning models composed of layers of interconnected units (or neurons) that transform input data and learn to recognize patterns through a training process.

Deep Neural Networks (DNNs) are a subset of NNs with multiple hidden layers, enabling them to model highly complex functions. Their depth allows them to capture complex patterns and abstract features, making them particularly effective for large-scale and high-dimensional data. DNNs have become indispensable in various applications, including computer vision (e.g., image recognition, and video analysis), natural language processing, automated medical diagnostics, and fraud detection, among others.

However, training DNNs requires large datasets, significant computational resources, and carefully fine-tuned algorithms [5,27]. As a result, trained DNNs have become valuable intellectual assets, making them attractive targets for attackers seeking to extract the model rather than invest in training their own.

In many cases, DNNs are deployed as cloud-based or online services, allowing users to interact with them without direct access to their internal parameters [25,1]. A natural question, therefore, is how well different DNNs resist attacks in this black-box setting, where an adversary can query the network using random or carefully chosen inputs—potentially adaptively—and exploit the observed outputs to reconstruct either the exact internal parameters or an approximation sufficient to construct a functionally equivalent network.

Extracting the parameters of a neural network from its black-box implementation has been extensively studied since 2016 [26,20,24,14,8,11,19,13]. The difficulty of the extraction problem varies depending

on what the adversary is allowed to observe in the output. The most realistic but challenging scenario for the attacker is when only the final label is available. This label indicates the most probable category for a given input, without exposing additional confidence scores or intermediate values. This setting is notably referred to as S1 in [7]. At the other end of the spectrum, the easiest scenario, called S5 in [7], allows the attacker to access the raw output of the neural network before the normalization phase, providing much richer information for reconstruction.

A significant step in model extraction research was made by Carlini et al. [8] in 2020 within the S5 scenario, specifically for networks using ReLU as the activation function, where they reframed the problem from a cryptanalytic perspective. They observed that DNNs share key similarities with block ciphers, including their iterative structure, alternating linear and non-linear layers, and the use in parallel of a small function (S-box in block ciphers, activation function in neural networks) to ensure non-linearity. Taking inspiration from cryptanalytic attacks on block ciphers, particularly differential cryptanalysis [4], Carlini et al. proposed a new two-step iterative approach for recovering a model's internal parameters. Their method decomposes the recovery process into two distinct phases: *signature extraction* and *sign extraction*. In the first step, the absolute values of the weights in a layer are reconstructed (signature extraction), followed by a second step where the correct signs are determined (sign extraction). This process is applied iteratively, proceeding layer by layer. This approach enabled them to successfully extract neural networks, including a model with $100,000$ parameters trained on the `MNIST` digit recognition task, using only $2^{21.5}$ queries and less than an hour of computation. However, their method for extracting the signs had an exponential complexity in the number of neurons, limiting its feasibility to small networks with typically very few (at most 3) layers.

In 2024, Canales-Martínez et al. [6] extended the work of Carlini et al. [8] by introducing several new algorithms for the sign-recovery step in ReLU-based neural networks. Their improved sign-extraction methods permitted them to extract the parameters of significantly larger networks than those considered in [8]. For instance, they reported successfully extracting the parameters of a neural network with 8 hidden layers of 256 neurons each—amounting to 1.2 million parameters—trained on the `CIFAR-10` dataset [16] for image classification across 10 categories. It is important to note, however, that their experiments assumed the absolute values of the weights had already been recovered using the method of Carlini et al. [8] and were not implemented as full end-to-end attacks.

More recently, Foerster et al. [13] conducted a detailed analysis of the signature and sign extraction procedures, implementing and benchmarking them on various ReLU-based neural networks. Their work revealed that, contrary to what was previously believed, the main practical bottleneck in these attacks lies in the signature extraction step introduced in [8]. Because of this, both [8] and [13] failed in practice to recover weight parameters beyond the third layer. Thus, although the authors of [6] reported successful extraction from deeper networks, this was only partially true, as the weights were supposed to have been extracted with [8]'s method. Indeed, as Foerster et al. [13] demonstrated, the signature extraction step remains a major limitation, preventing all these methods from progressing beyond the early layers.

As of today, no algorithm is known to recover efficiently a neural network's parameters beyond the third layer[1]. This highlights the need for significant improvements in the signature recovery process to enable practical attacks on deeper networks.

Our Contributions

In this work, we revisit the signature extraction algorithm of [8], which serves as the basis for the most important parameter extraction attacks against NNs [8,6,9,7]. We start by carefully analyzing its limitations and highlighting the main problems that reduce its effectiveness. More specifically, our analysis reveals two major issues:

Impact of deeper layers on signature merging. Partial signature merging is a key step in signature recovery. While the impact of deeper layers on this process was previously considered negligible, we show that in deep neural networks, they introduce significant interference. This added noise often causes neurons to be misattributed to the wrong layer during merging.

[1] The authors of [6] report having successfully extracted the 4th layer of a single network. However, in that network, this 4th layer is very contractive and in this setting, the issues identified in our work do not arise.

Rank deficiency. In deeper layers, the system of equations collected by the attacker to recover a neuron's weights may be rank-deficient, leading to an erroneous neuron recovery.

In a second step, we propose concrete algorithmic solutions to overcome both identified issues. We implemented and evaluated these solutions on neural networks trained on different datasets. Our results demonstrate, for the first time, that it is possible to extract nearly all weights from deep networks. In particular, while previous state-of-the-art signature extraction methods could not recover in practice layers beyond the third layer, our approach successfully extracts neurons from networks with up to eight hidden layers. We also applied our methods to even deeper networks with sixteen hidden layers and but we encountered runtime limitations that prevented full recovery.

2 Related Work

Deep neural network model extraction is an old and well-studied problem. The earliest work dates back to 1994, when Fefferman [12] proved that the output of a sigmoid network uniquely determines its architecture and weights, up to trivial equivalences. Later, in 2005, Lowd and Meek [18] introduced new algorithms for reverse engineering linear classifiers and applied these techniques to spam filtering, marking one of the first practical extraction attacks. The field saw renewed interest in 2016 with the work of Tramèr et al. [26], who demonstrated attacks on deployed machine learning models accessible through APIs that output high-precision confidence values (also referred to as scores), in addition to class labels. By exploiting these confidence scores, they successfully mounted attacks on various model types, including logistic regression and decision trees.

For non-linear models, [26] and later [23] introduced the notion of *task accuracy extraction*, where the goal is to build a model that performs well on the same decision task, achieving high accuracy on predictions—without necessarily reproducing the exact outputs of the original model. This is different from *functionally equivalent extraction*, where the objective is to replicate the original model's outputs on all inputs, regardless of the ground truth. A related, but narrower concept is *fidelity extraction*, which focuses on replicating the model's predictions over a specific input distribution. Jagielski et al. [15], who introduced this taxonomy, argued that learning-based approaches like [26,23] are fundamentally limited when it comes to achieving functionally equivalent extraction.

Early attempts for functionally equivalent extraction relied on access to internal gradients of the model [20], side-channel information [3], or were limited to extracting only a small number of layers [15]. A turning point came with the work of Carlini et al. [8], who showed strong connections between the extraction of ReLU-based neural networks and the cryptanalysis of block ciphers. More precisely, they exhibited in this work a differential-style attack capable of recovering significantly larger models than prior approaches. This attack can be divided into two parts: signature extraction (recovering the absolute value of the internal weights) and sign extraction (recovering the signs of the internal weights). This method permitted the authors of [8] to successfully reverse-engineer model parameters trained on random data with at most three hidden layers.

On the other hand, the sign recovery process of [8] was identified to be very time-consuming (exponential in the number of neurons) and therefore impractical for deeper models trained on standard datasets. To overcome this limitation, Canales-Martinez et al. [6] proposed a polynomial-time sign-extraction algorithm, the *neuron wiggle*.

Recently, Foerster et al. [13] sped up the sign recovery and conducted a thorough performance evaluation of the signature [8] and sign [6] extraction methods. They showed that the signature extraction algorithm does not give satisfying results for deeper layers, even if ran for a much longer time.

In parallel with these efforts to improve performance under full access to confidence scores, other members of the research community have turned their attention to the more realistic "hard-label" scenario [10,7], where the DNN outputs only the predicted class label, corresponding to the highest confidence score, while the actual confidence values remain hidden. Finally, we emphasize that neither [8,13] nor [10,7] managed to extract weights deeper than the third layer (see previous footnote).

We refer to [22] for a recent survey of the field.

3 Preliminaries

In this section, we introduce important definitions and notations (Section 3.1), followed by assumptions regarding the attack setting (Section 3.2 and Section 3.3) and finally an overview of the original attack of [8] (Section 3.4).

3.1 Notations and Definitions

This paper models neural networks as parametrized functions whose parameters are the unknowns we aim to extract. Our results do not depend on how neural networks are trained or applied. As a result, no prior knowledge about them is required to understand the attack.

A neural network consists of fundamental units called neurons, which are organized into layers and connected to other neurons in both the previous and next layers. Each neuron has an associated weight vector for its incoming neurons from the previous layer, along with a bias term that influences its output. Following the notations and definitions from [8,6], we now present several central definitions related to neural networks. A simple example to illustrate the definitions can be found in Appendix A.

Definition 1 (r-deep neural network). *An r-*deep fully connected neural network *of architecture $[d_0, \cdots, d_{r+1}]$ is a function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{r+1}}$ composed of alternating linear layers $\ell^{(i)} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ and a non-linear activation function σ acting component-wise such that: $f = \ell^{(r+1)} \circ \sigma \circ \cdots \circ \sigma \circ \ell^{(2)} \circ \sigma \circ \ell^{(1)}$, where $\ell^{(i)}(\boldsymbol{x}) = \boldsymbol{A}^{(i)}(x) + b^{(i)}$; $\boldsymbol{A}^{(i)} \in \mathbb{R}^{*^{d_i \times d_{i-1}}}$, $b^{(i)} \in \mathbb{R}^{d_i}$. We call:*

- *r: the number of layers (or* depth *of the network).*
- *d_i: the number of neurons in the i-th layer (or* width *of layer i).*
- *$\ell^{(i)}$: the i-th linear layer function.*
- *$\boldsymbol{A}^{(i)}$: the i-th linear layer weight matrix.*
- *$b^{(i)}$: the i-th linear layer bias vector.*

To extend the notations, the architecture $[d_0, \ldots, d_{r+1}]$ can also be written as $d_0 - \cdots - d_{r+1}$. For consecutive layers with the same dimension, an exponential notation can be used for compactness, e.g., $20 - 10^{(3)} - 1$ represents the architecture $20 - 10 - 10 - 10 - 1$.
As in [8,6,13], this research only considers fully connected neural networks using the widespread ReLU activation function [21] applied component-wise. An example of such a network is depicted in Figure 1. The structure of a fully connected neural network resembles that of substitution-permutation networks (SPNs) such as the AES [2]. The component-wise application of the activation function is analogous to the use of S-boxes in SPNs, hence the analogy with cryptanalysis introduced in [8].

Definition 2 (ReLU neural network). *We say that a neural network f defined as above is a* ReLU neural network *if its non-linear activation function σ is*

$$\begin{aligned}\sigma(v) &= (\mathit{ReLU}(v_1), \mathit{ReLU}(v_2), \ldots, \mathit{ReLU}(v_n)) \\ &= (\max(v_1, 0), \max(v_2, 0), \ldots, \max(v_n, 0))\end{aligned}$$

for $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$.

In the following, we define a reduced-round neural network $F^{(i)}$ as a function $F^{(i)} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{i+1}}$ given by:

$$F^{(i)} = \ell^{(i)} \circ \sigma \circ \cdots \circ \sigma \circ \ell^{(2)} \circ \sigma \circ \ell^{(1)}.$$

$F^{(i)}$ shares the same linear transformations and activation functions as the original function f, up to layer i. It will be used to track the transformation of an input vector as it goes through f. Observe that $F^{(r+1)} = f$.

More formally, the *k-th neuron* of layer i is defined as the function $\eta_k^{(i)}(x) = A_k^{(i)} \cdot x + b_k^{(i)}$, where $A_k^{(i)}$ is the k-th row of $\boldsymbol{A}^{(i)}$ and $b_k^{(i)}$ is the k-th coordinate of $b^{(i)}$.

A central notion in the analysis of [8] is that of *critical points*—input points at which a specific neuron becomes inactive (i.e. when its output is zero). These points play a crucial role, as they enable the extraction of the corresponding neuron's weights through carefully chosen queries. A formal definition is provided below.

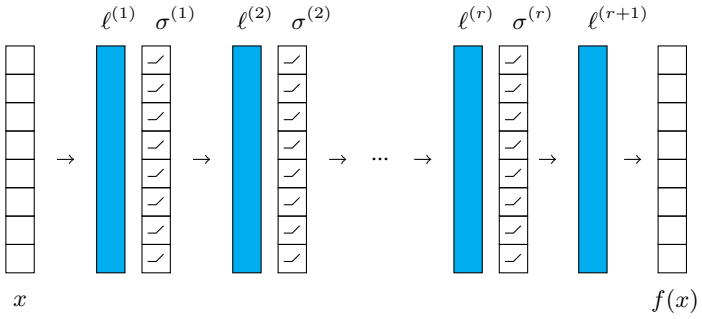

Fig. 1: Fully connected ReLU neural network

Definition 3 (critical point). *An input point $x \in \mathbb{R}^{d_0}$ is called a* critical point *if there exists an index pair (i, k) such that:*

$$[F^{(i)}(x)]_k = 0, \quad \textit{for } i \in \{1, \dots, r\}, \quad k \in \{1, \dots, d_i\}.$$

where $[\cdot]_k$ refers to the k-th entry of a vector. Equivalently, this holds if $\eta_k^{(i)} \circ \sigma \circ F^{(i-1)}(x) = 0$. For this reason, we say that x is a critical point *of neuron $\eta_k^{(i)}$.*

The activation pattern for a given input x is the set of all (active) neurons that contribute to the output $f(x)$. A formal definition is given below:

Definition 4 (activation pattern). *The* activation pattern *of x through f is the set*

$$\mathcal{S}_f(x) := \{\eta_k^{(i)} \mid [F^{(i)}(x)]_k \geq 0, \quad 1 \leq i \leq r, 1 \leq k \leq d_i\}.$$

If $\eta_k^{(i)} \in S_f(x)$ then we say that neuron $\eta_k^{(i)}$ is active, otherwise we say that it is inactive (for x and f).

Definition 5 (polytope). *The polytope of x is the set:*

$$\mathcal{P}_x := \{x' \in \mathbb{R}^{d_0} \mid \mathcal{S}_f(x) = \mathcal{S}_f(x')\}$$

Therefore, $\mathcal{P}_x$ represents the region of the input space consisting of all points x' close enough to x for the network to activate the same subset of neurons for x' as it does for x. Later, we will justify the use of the term *polytope* to describe this space.

Lemma 1 (local affine network). *First, for $x \in \mathbb{R}^{d_0}$, the network f is affine on $\mathcal{P}_x$, meaning that there exists (Γ_x, γ_x) such that $\forall x' \in \mathcal{P}_x$, we have $f(x') = \Gamma_x x' + \gamma_x$. We note:*

- *f_x the function $x' \mapsto \Gamma_x x' + \gamma_x$.*
- *$F_x^{(i)}$ the function $x' \mapsto \Gamma_x^{(i)} x' + \gamma_x^{(i)}$.*

Second, it follows that if x is not a critical point, there exists $\epsilon > 0$ such that for all $x' \in \mathcal{B}(x; \epsilon)$,

$$f(x') = f_x(x') \quad \textit{and} \quad \forall i \in \{1, \dots, r+1\}, \quad F^{(i)}(x') = F_x^{(i)}(x'),$$

where $\mathcal{B}(x; \epsilon) \subset \mathbb{R}^{d_0}$ is the ball of radius ϵ centered at x.

Though ReLUs are not linear, they are piecewise linear. This means that when moving near a non-critical point, all ReLUs remain linear. Recall that critical points are solutions of an equation of the form $A_k^{(i)} \cdot F^{(i-1)}(x) + b_k^{(i)} = 0$. Thus, they form hyperplanes of dimensions $d_0 - 1$, which partition the input space into these different affine regions, hence referred to as *polytopes*.

We denote by $I_x^{(i)}$ the diagonal matrix representing which neurons in layer i are active for the input x: the k-th coefficient of the diagonal is 1 if the k-th neuron in layer i is active, and 0 otherwise. Thus,

$$\begin{aligned} F^{(i)}(x) &= I_x^{(i)}(A^{(i)} \cdots (I_x^{(2)}(A^{(2)}(I_x^{(1)}(A^{(1)}x + b^{(1)})) + b^{(2)}) \cdots + b^{(i)}) \\ &= I_x^{(i)} A^{(i)} \cdots I_x^{(2)} A^{(2)} I_x^{(1)} A^{(1)} x + \gamma_x^{(i)} = \Gamma_x^{(i)} x + \gamma_x^{(i)} \end{aligned}$$

If x and x' are two points in the same polytope, then $\forall i, I_x^{(i)} = I_{x'}^{(i)}$ implying that $\Gamma_x^{(i)} = \Gamma_{x'}^{(i)}$ and $\gamma_x^{(i)} = \gamma_{x'}^{(i)}$, i.e., that $F_x^{(i)} = F_{x'}^{(i)}$.

3.2 Adversarial Resources and Goal

We consider two parties in this model extraction attack: an oracle $\mathcal{O}$ and an adversary. The adversary generates queries x and sends them to the oracle, which then responds with the correct output $f(x)$.

Adversarial resources. We make the following assumptions regarding the target neural network and the attacker's capabilities. These are the same assumptions as in [8,6,13]:

- **Fully connected ReLU network**. f is a fully connected ReLU neural network.
- **Known architecture**. The attacker knows the architecture of the target neural network f, meaning she knows the depth of f and the width of each layer.
- **Unrestricted input access**. The attacker can query the network on any input $x \in \mathbb{R}^{d_0}$.
- **Raw output access**. The oracle returns the complete raw output $f(x)$, with no post-processing.
- **Precise computations**. The oracle computes $f(x)$ using 64-bit arithmetic.

Adversarial goal. The objective of the extraction is not to exactly replicate the target network, but rather to achieve what is known as an (ϵ, δ)-*functionally equivalent extraction* [8]. We slightly change the definition to normalise the difference between the target and the extracted network.

Definition 6 (functionally equivalent extraction). *We say that two models f and $\hat{f}$ are* (ϵ, δ)-functionally equivalent *on an input space $S \subset \mathbb{R}^{d_0}$ if* $\forall x \in S$, $\mathbb{P}(|\frac{\hat{f}(x)-f(x)}{f(x)}| \leq \epsilon) \geq 1 - \delta$.

3.3 On Functionally Equivalent Extraction

We note four key differences between the extracted and target networks and justify that they will not affect the (ϵ, δ)-assessment.

Permutation of neurons. During the attack, we will extract a neuron from a hidden layer, but we will not know to which row (neuron in the layer) it corresponds. For example, if we denote the first extracted neuron as $\hat{A}_1^{(i)}$, we do not know if $\hat{A}_1^{(i)} = A_1^{(i)}$ or if $\hat{A}_1^{(i)} = A_2^{(i)}$, etc. We are therefore permuting the rows of $A^{(i)}$. However, when extracting the next hidden layer $\hat{A}^{(i+1)}$, the process naturally permutes the columns to match the permuted rows of $\hat{A}^{(i)}$, since $\hat{A}^{(i)}$ (rather than $A^{(i)}$) is used in the extraction. Therefore, this permutation does not compromise the overall extraction process over the layers. For the last output layer, we must directly align our extracted matrix with the output of f as this ensures that this last recovered layer has the correct row order. We can conclude that the permutation of extracted neurons in a hidden layer affects neither ϵ nor δ.

Scaling of neurons. When extracting a neuron $\eta_k^{(i)}$ (row k from $A^{(i)}$), its values are scaled by a constant $c_k^{(i)}$. This scaled row is referred to as the *signature* of $A_k^{(i)}$. Since the extraction is performed on $\hat{A}^{(i)}$ rather than on $A^{(i)}$, every corresponding column in $A^{(i+1)}$ must be adjusted by dividing its values by $c_k^{(i)}$ to match $\hat{A}^{(i)}$. Even though $\hat{a}_{j,k}^{(i+1)} = \frac{a_{j,k}^{(i+1)}}{c_k^{(i)}} \cdot c_j^{(i+1)}$ for any $a_{j,k}^{(i+1)} \in A^{(i+1)}$, we still have,

$$\hat{\Gamma}^{(i+1)} = \begin{pmatrix} c_1^{(i+1)} & 0 & \cdots & 0 \\ 0 & c_2^{(i+1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{d_{i+1}}^{(i+1)} \end{pmatrix} \cdot \Gamma^{(i+1)}.$$

Once again, because the last output layer must match the output of the network, we have $\hat{a}_{j,k}^{(r+1)} = \frac{a_{j,k}^{(r+1)}}{c_k^{(r)}} \cdot 1$ and therefore $\hat{\Gamma} = \hat{\Gamma}^{(r+1)} = \Gamma^{(r+1)} = \Gamma$. Thus, here again, scaling does not affect ϵ and δ.

Always-on/off neurons. Some neurons do not have critical points within the space of queries the attacker will make. They are either always active (on) or always inactive (off) in this input space. The attacker relies on critical points to identify hyperplanes and detect the existence of neurons. As such, always-on and always-off neurons cannot be detected or recovered through extraction. Always-off

neurons can safely be ignored as they do not contribute to the network. Always-on neurons do not induce additional critical hyperplanes in $\mathbb{R}^{d_0}$ and make a linear map f on the input space. The extraction naturally integrates this map when recovering the last layer of f. The authors of [6] reported that no always-on neurons were found on trained neural networks, but they did find instances of always-off neurons. In some cases, it can be relevant to extract the network on a small interesting subset S of $\mathbb{R}^{d_0}$, where some neurons might be always-on. For example, if the target is a neural network tasked with image recognition, such a subset can be the hyperplane corresponding to black-and-white pictures. Extracting the neural network only on this hyperplane will make the attack less costly in time and in queries.

Dead weights. In some cases, some weights can be set to 0 without affecting the effectiveness of the extracted network $\hat{f}$. We refer to these weights as *dead weights*. A key instance comes from always-off and always-on neurons when we assume the correct extraction of all previous layers. Specifically, if $\eta_k^{(i)}$ is always on or always off, the weights on the k-th column of $A^{(i+1)}$ are dead weights as their contribution is absorbed by the last layer as explained above. We will come back to them in Section 5.

3.4 Overview of the Existing Attack

In this section, we recall the signature extraction algorithm from [8]. The aim, as discussed earlier, is to extract an (ϵ, δ)-functionally equivalent network. The adversarial goals and the resources required for the extraction are outlined in Section 3.2. The extraction is performed layer by layer: first, we recover the parameters of layer 1, then use them to reconstruct layer 2, and so on. We now explain the process of recovering layer i, assuming that the first $i-1$ layers of the target network f have been correctly extracted.

The recovery of each layer consists of two parts. The first part is a signature recovery (following [8]), which recovers each matrix row $A_k^{(i)}$ up to a scaling factor. The second part is sign recovery (following [13]'s tweak of [6]) which determines the sign of the scaling factor for each neuron. As our focus is on signature recovery, we refer the reader to [13] and [6] for a description of the sign recovery.

Signature recovery is carried out in five main steps: searching critical points, recovering partial signatures, merging partial signatures, finding missing entries in the signatures, and computing the bias.

Random search for critical points. In each polytope, the network behaves affinely, meaning that the derivative of f remains constant within the same polytope. Therefore, critical points can be identified when a change in the derivative occurs, indicating that a critical hyperplane has been crossed. To find these critical points, we apply a binary search along random lines in the input space. See Appendix B for a detailed explanation.

Partial signature recovery. When we cross a neuron's critical hyperplane, we move, depending on the sign of the neuron, from a region (polytope) where the neuron does not contribute to the network's output to one where it does. All other neurons remain unchanged. Therefore, by making queries close to a critical point, we can construct a system of equations that isolates the contribution of the neuron in question, allowing us to recover its parameters up to a sign. We now describe this process in more detail.

We define the second-order differential operator of f along direction Δ as $\partial^2_\Delta f(x) := f(x+\epsilon\Delta) + f(x-\epsilon\Delta) - 2f(x)$. Suppose the target layer is layer i. We assume that the previous layers have been extracted and thus that $\Gamma_x^{(i-1)}$ is known for all inputs x.

Lemma 2 ([8]). *Let x be a critical point for a neuron $\eta_k^{(i)}$ in layer $i \in \{1, \dots, r\}$, and assume that x is not a critical point for the other neurons. Further, let $\Delta \in \mathbb{R}^{d_0}$. Then:*

$$\partial^2_\Delta f(x) = c_k^{(i)} \left| (\Gamma_x^{(i-1)} \Delta) \cdot A_k^{(i)} \right|$$

where $c_k^{(i)} \in \mathbb{R}$ is a constant.

Proof. See Appendix C.1.

By using Lemma 2 in two directions, Δ and Δ_0, we obtain

$$\frac{\partial^2_{\Delta} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{\left|(I_x^{(i-1)}\Delta) \cdot A_k^{(i)}\right|}{\left|(I_x^{(i-1)}\Delta_0) \cdot A_k^{(i)}\right|}.$$

Then, by comparing the absolute values between

$$\frac{\partial^2_{\Delta} f(x) + \partial^2_{\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)} \text{ and } \frac{\partial^2_{\Delta+\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)},$$

we can eliminate the absolute value (see Appendix C.2), obtaining,

$$\frac{(I_x^{(i-1)}\Delta) \cdot A_k^{(i)}}{(I_x^{(i-1)}\Delta_0) \cdot A_k^{(i)}}.$$

By taking enough directions Δ in the input space, we can solve for $A_k^{(i)}$ up to a constant, reconstructing the signature (see Appendix C.3). Since $I_x^{(i-1)} = I_x^{(i-1)} A^{(i-1)} I_x^{(i-2)}$, ReLUs on layer $i-1$ are blocking some coefficients for all directions Δ we take around x. Therefore, instead of extracting a full (scaled) matrix row, for example $(a_1, a_2, a_3, a_4, a_5)$, we can only retrieve a partial signature, say, $(0, a_2, a_3, 0, a_5)$.

Merging signatures into components. To reconstruct the full signature, we need to merge partial signatures obtained from different critical points of the same neuron $\eta_k^{(i)}$. Each recovered signature from critical points of $\eta_k^{(i)}$ is $A_k^{(i)}$ with some missing entries, scaled to an unknown factor. Assuming that no two rows of the matrix are identical, merging two signatures requires only checking whether all their shared non-zero weights are proportional. For example, consider two signatures $(\lambda a, \lambda b, 0, \lambda d, 0)$ and $(\Lambda a, \Lambda b, 0, 0, \Lambda e)$ where λ and Λ are constants. If $\frac{\lambda a}{\lambda b} = \frac{\Lambda a}{\Lambda b}$, they can be merged to get $(a, b, 0, d, e)$, up to a constant (see Figure 2). The resulting merged row is called a *component*. The *size of a component* refers to the number of critical points whose partial signatures have merged to get the component.

If a component has a size greater than 2, we consider it to belong to the target layer, as it is highly unlikely for signature extraction to yield consistent results when applied to critical points from deeper layers. We repeat the process described above — randomly searching for critical points, extracting partial rows, and merging signatures — until we obtain at least d_i components of size greater than 2, or until a predefined threshold on the number of critical points is reached (to prevent infinite loops caused by always-off or always-on neurons). If the number of components exceeds d_i, the number of neurons in layer i, we retain only the largest ones.

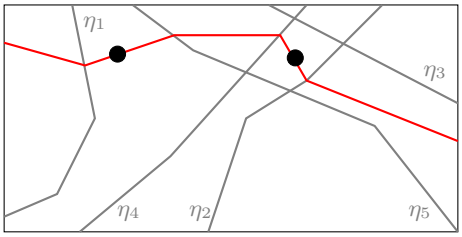

Fig. 2: Network where neurons on the previous layer are labelled in grey on their active side. In red is the neuron we aim to find. The critical points on the left and right yield respectively $(\Lambda a, \Lambda b, 0, 0, \Lambda e)$ and $(\lambda a, \lambda b, 0, \lambda d, 0)$. We can infer $(a, b, 0, d, e)$ up to a constant, despite the hyperplane in red never entering a polytope where η_1, η_2, η_4 and η_5 are active.

Targeted search for critical points. At this stage of the attack, we know that all selected components correspond to neurons on the target layer. However, components recovered from random critical points

often have weights missing, even after merging numerous partial signatures. To complete the partial rows, a more targeted search for critical points is necessary. We start from one of the critical points within the component. As we cross different hyperplanes associated with neurons in the previous layer, we activate different weights of the target neuron, ultimately enabling its full reconstruction.

For example, in Figure 2, we would follow the red hyperplane as it bends across the grey hyperplanes, retrieving enough critical points to reconstruct the full signature of our neuron. In higher dimensions, we follow directions that are more likely to trigger the weights we have not found yet. The exact procedure is described in [8] and further improved in [13].

Recovering the bias. Once we have recovered the signature of $\eta_k^{(i)}$, $c_k^{(i)} A_k^{(i)}$, it suffices to pick a critical point x of the component to compute the corresponding scaled bias $c_k^{(i)} b_k^{(i)}$ using the equation

$$(c_k^{(i)} A_k^{(i)}) \cdot F^{(i-1)}(x) + c_k^{(i)} b_k^{(i)} = 0.$$

Last layer recovery. The oracle returns the complete raw outputs of f and the last output layer is a linear layer. Therefore its extraction is straightforward. No additional query is needed for extraction, as we can reuse previously queried critical points to build the linear system.

4 Improving the Signature Recovery

To the best of our knowledge, no prior work has successfully applied signature extraction on deep enough neural networks. Specifically, according to Table 1 in [8], the deepest model the authors of this paper attempted to attack had an architecture of $40 - 20 - 10 - 10 - 1$, consisting of only 3 hidden layers. On the other hand, Table 2 in [13] shows that they attempted to attack a model with an architecture of $784 - 16^{(8)} - 1$, which had 8 hidden layers. However, despite running the attack for over 36 hours, they failed to recover the fourth hidden layer. Although [6] demonstrated that their sign recovery was effective across all layers of a complex $3072 - 256^{(8)} - 1$ model, they did not conduct experiments on signature recovery.

Motivated by these limitations, we analyze the signature extraction method and identify two challenges preventing its successful application on deeper layers. We then propose solutions to overcome them. The left part of Figure 3 illustrates the signature extraction workflow and highlights where these challenges arise.

We provide a brief overview of each failure. First, the signature extraction relies on solving a system of equations to recover the weights of the target neuron associated with all active neurons in the previous layer. However, as we go deeper into the network, the rank of this system might not match the number of active neurons on the previous layer. This mismatch can lead to an underdetermined system, resulting in incorrect signature extraction. Second, the component selection of the largest d_i components of size greater than 2 is incorrect. Our experiments show that this strategy is only valid for shallow networks with comparatively fewer critical points on deeper layers. Figure 3 gives an outline of our improvements.

4.1 Identifying Deeper Merges

While going through the signature extraction procedure, we made the crucial assumption that x is a critical point of a neuron on the layer we are targeting. Since we are trying to extract that layer, we cannot verify this assumption. We might be extracting a neuron of a deeper layer. The authors of [8] claim that it is exceedingly unlikely that signatures extracted from critical points on a deeper layer can be merged into a component. They conclude that a component of size greater than two is on the target layer. In this section, we explain why signatures from deeper layers can indeed be merged, discuss why this can be a problem, and propose potential solutions to address this issue.

When we extract from two critical points x_1 and x_2 of the same neuron in a deeper layer $i + t$ using its preceding layer $i + t - 1$, we solve for y_1 and y_2 in the respective systems below and observe that y_1 and y_2 can be merged:

$$\partial^2_\Delta f(x_1) = (\Gamma_{x_1}^{(i+t-1)} \Delta) \cdot y_1$$
$$\partial^2_\Delta f(x_2) = (\Gamma_{x_2}^{(i+t-1)} \Delta) \cdot y_2$$

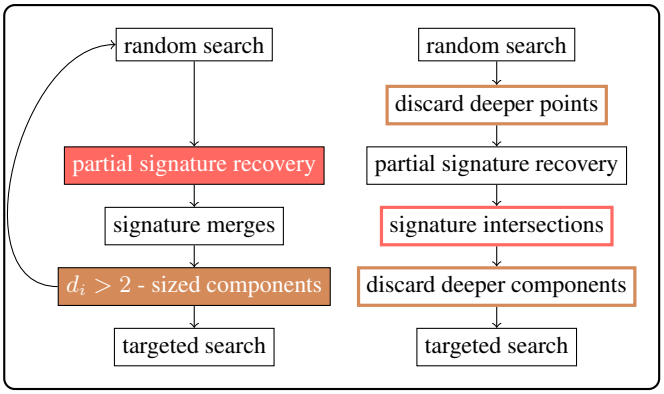

Fig. 3: Left: Original attack from [8]. Right: Proposed improvements. Two error-inducing steps in the original attack are coloured on the left. Improvements match the colour of the step they address.

Let's write A_{x_1} the matrix $\ell^{(i+t-1)} \circ I_{x_1}^{(i+t-2)} \circ \cdots \circ I_{x_1}^{(i)} \circ \ell^{(i)}$ and A_{x_2} the matrix $\ell^{(i+t-1)} \circ I_{x_2}^{(i+t-2)} \circ \cdots \circ I_{x_2}^{(i)} \circ \ell^{(i)}$. Thus, we can rewrite these two systems with A_{x_1} and A_{x_2},

$$\partial^2_\Delta f(x_1) = [(A_{x_1} \circ \Gamma^{(i-1)}_{x_1})\Delta] \cdot y_1$$
$$\partial^2_\Delta f(x_2) = [(A_{x_2} \circ \Gamma^{(i-1)}_{x_2})\Delta] \cdot y_2$$

Now suppose that x_1 and x_2 have the same activation pattern - meaning they set the same neurons as active neurons per layer - between layers i and $i+t-1$. In this case, $A_{x_2} = A_{x_1}$, which we write as A. Therefore, our systems are as follows.

$$\partial^2_\Delta f(x_1) = [(A \circ \Gamma^{(i-1)}_{x_1})\Delta] \cdot y_1$$
$$\partial^2_\Delta f(x_2) = [(A \circ \Gamma^{(i-1)}_{x_2})\Delta] \cdot y_2$$

These systems respectively yield the same solutions as:

$$\partial^2_\Delta f(x_1) = (\Gamma^{(i-1)}_{x_1}\Delta) \cdot (A^\top y_1)$$
$$\partial^2_\Delta f(x_2) = (\Gamma^{(i-1)}_{x_2}\Delta) \cdot (A^\top y_2).$$

Since y_1 and y_2 can merge, so can $A^\top y_1$ and $A^\top y_2$. These systems correspond exactly to the partial signature extraction from x_1 and x_2 when extracting layer i. This is why the partial signatures extracted from x_1 and x_2 can merge into a component even though they are not on the target layer. We call all these unwanted additional components from deeper layers *noise components*.

How does this noise behave? First, the smaller the width of the network, the larger the noise, as critical points from deeper neurons are partitioned into fewer components. Second, the deeper the network, the larger the noise, as critical points on the target layer represent a smaller fraction of the critical points we find. We believe this is one of the reasons why previous works still managed to perform the extraction on shallow networks.

In addition, in both [8] and [13]'s implementations, the selected neurons are the d_i *largest* components of size greater than two. When using this criterion, noise becomes a problem when the size of a noise component becomes larger than the size of a component corresponding to a neuron on the target layer.

We illustrate the issue of deeper critical points with an example slightly deeper than what previous works managed to recover: using $3,000$ critical points we extract the 4th layer of a model trained on the `MNIST` dataset with architecture $784 - 8^{(8)} - 1$. Due to space constraints, only the largest 16 components are displayed. As shown in Figure 4, four of the first eight components recovered by the original signature extraction belong to deeper layers. This shows that, in deeper networks, the size of a component is not a sufficient indicator to identify if the neuron is on the target layer.

Discarding points on deeper layers. To discard points on deeper layers, we first recycle and improve an algorithm from [8] that was only used in the context of the targeted search. The process involves 1)

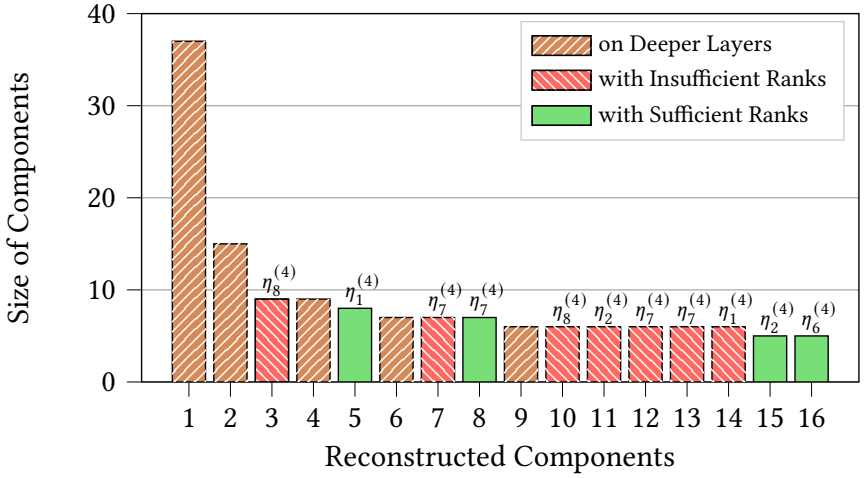

Fig. 4: Merging results from the original attack on layer 4 of a $784 - 8^{(8)} - 1$ `MNIST` model with $3,000$ critical points. Each component on the target layer is labelled with its associated neuron on top. Larger component size does not necessarily indicate that the component belongs to the target network.

computing 100 distinct intersection points between the extracted hyperplane and the extracted network for the critical point under test, and 2) verifying on the target neural network whether these points lie on the extracted hyperplane. One intersection point not belonging to the extracted hyperplane indicates that the hyperplane has bent on a neuron that has not yet been extracted, and hence that the extracted hyperplane is on a deeper layer. This test is not an if-and-only-if condition, as a hyperplane not breaking does not guarantee that it corresponds to a neuron on the target layer (see Figure 6). We apply this test to every critical point we find. This reduces the noise, but it is not sufficient. As exemplified in Figure 5, though we remove one component from deeper layers, four large such components remain.

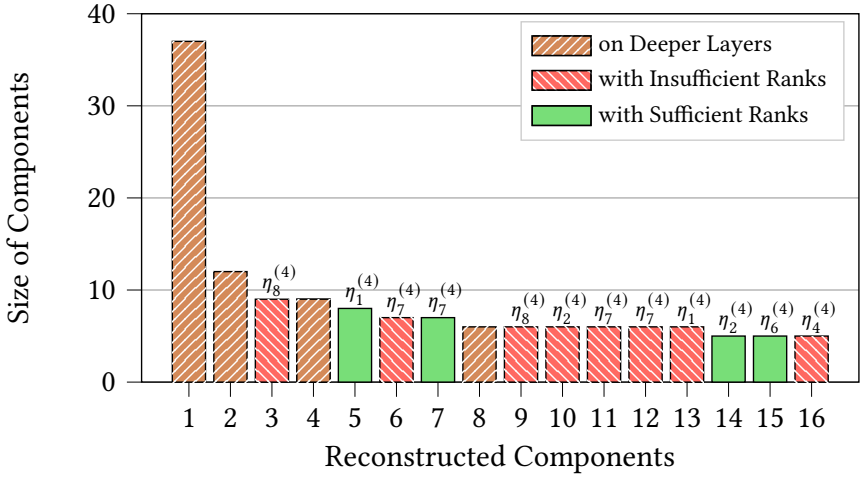

Fig. 5: Merging results after discarding critical points on deeper layers. Same experimental setup as in Figure 4. Components on deeper layers either disappear or have a smaller size.

Discarding deeper components. We need to introduce additional discrimination techniques to reduce the interference of deeper layers. While the strategy above aimed at discarding points on deeper layers, we now turn to components. We use three criteria. First, we know that critical points that merge usually

belong to the same neuron, regardless of whether they are in the target layer or a deeper layer. Therefore, the more critical points identified from deeper layers by the above test that merge with our candidate component, the more likely it is that this component belongs to a deeper layer. For this reason, we compute for each component a noise ratio

$$\tau = \frac{\text{\#merges with deeper critical points}}{\text{size of the component}}.$$

Second, because merges of deeper critical points come from critical points with the same partial activation pattern, those deeper components have a harder time obtaining a diverse set of critical points to yield a neuron with a very small number of entries unrecovered. Finally, because of the constraint on the activation pattern of deeper merges, components on deeper layers tend to be smaller in size. While none of these criteria are if-and-only-if, combining them yields satisfying results. First, we remove components with a size smaller than a certain fraction of the largest component (we found that in practice, 0.1 was sufficient for networks with eight hidden layers). Second, we discard components that either have a τ above a certain threshold (we use 0.1 for networks with width 8 and 0.2 for networks with width 16) or a large number of entries unrecovered (at least half of them) if their $\tau > 0$.

It's worth noting that making a strict association between deeper critical points and incorrect components could lead to mislabelling. For example, critical points of a neuron in a deeper layer $i + t$ can merge with those of neuron $\eta_k^{(i)}$ on the target layer i, if they cause $\eta_k^{(i)}$ to be the only active neuron on the target layer (proof in Appendix D).

This case is very rare, even for small networks, but it might still happen. We encountered two such neurons in our experiments. The extracted component will have the right weights, but we might discard it because of a high τ. We check for this phenomenon on neurons with a low τ (neurons we are almost sure are on the target layer): first we finish the extraction of those neurons, we could compute their sign using the neuron wiggle and if a critical point is on the inactive side of all those neurons, we do not count it towards its component's τ.

On a side note, when this happens, the sign of all neurons on layer i can be recovered immediately with no queries or computations as we have an input x^* for which all neurons on the layer must be inactive (first order derivatives in $\mathcal{P}_{x^*}$ are equal to zero). We must have $\eta_k^{(i)} \cdot x^* + b_j^{(i)} < 0$, for all neurons on the layer, so we choose the sign of each $\eta_k^{(i)}$ to match this condition.

The graph illustrating the effectiveness of discarding deeper components is presented later in Figure 9, as the attack first needs to deal with another kind of limitation.

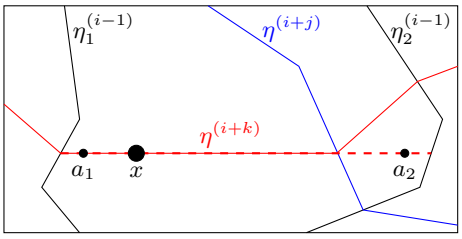

Fig. 6: Identifying if x is on the target layer. $0 \leq j < k$. By finding that a_2 is not a critical point, we can infer that the hyperplane we extracted from x (- - -) broke on a layer $i + j$ we have not extracted (——). Therefore, x cannot be on layer i. However, finding that a_1 is a critical point does not give any information regarding x's layer.

4.2 Increasing Rank with Subspace Intersections

When extracting the signature, our goal is to obtain a partial signature y_x up to an unknown constant from a critical point x (see Section 3.4). As a result, the solution space should be of dimension 1. To

determine y_x, we solved the equation $(\Gamma_x^{(i-1)}\Delta) \cdot y_x = \partial_\Delta^2 f(x)$ by selecting a sufficient number of random directions Δ. The partial row obtained consists of coefficients corresponding to the active neurons in layer $i-1$ for the given input x. We denote by:

- $s_x^{(i-1)}$: the number of active neurons in layer $i-1$,
- $r_x^{(i-1)} := \text{rank}(\Gamma_x^{(i-1)})$.

If $s_x^{(i-1)} > r_x^{(i-1)}$, the system is underdetermined and cannot be solved. This issue arises when there are fewer active neurons in a previous layer $k < i-1$ than in layer $i-1$ (see Figure 7), as indeed:

$$\Gamma_x^{(i-1)} = I_x^{(i-1)} A^{(i-1)} \cdots I_x^{(1)} A^{(1)}$$

and thus,

$$\text{rank}(\Gamma_x^{(i-1)}) = \min(\{\text{rank}(I_x^{(k)})\}_{1 \leq k \leq i-1}) \neq \text{rank}(I_x^{(i-1)})$$

If X denotes the random variable representing the number of active neurons per layer, then $\mathbb{P}(r_x^{(i-1)} < s_x^{(i-1)}) = 1 - \mathbb{P}(X \geq s_x^{(i-1)})^{i-1}$. Because of the $i-1$ exponent, it appears infeasible to extract deeper layers beyond the first few. **Signature intersections.** If the rank is insufficient, we no longer have a

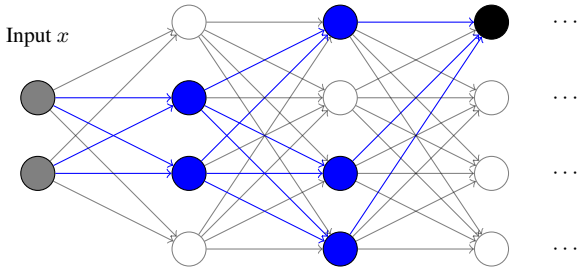

Fig. 7: Solving for a neuron's weights with certain critical points can lead to underdetermined systems. Target neuron is in black. Active neurons are in blue. The rank of $\Gamma_x^{(1)}$ is 2, which limits the rank of $\Gamma_x^{(2)}$ to 2 while we expect 3.

unique solution but rather a set of solutions. For a critical point x in the layer i, the solution space is given by $\mathcal{S}_x = L_x + \ker(\Gamma_x^{(i-1)})$, where L_x is a solution of the solution space and $\ker(M)$ represents the kernel space of a matrix M. In [8], a method was proposed to merge two partial signatures. Here, we extend this approach to merge two solution spaces.

Suppose we have another critical point x' with solution space $\mathcal{S}_{x'} = L_{x'} + \ker(\Gamma_{x'}^{(i-1)})$. Since we recover rows up to a scalar factor, we seek to compute the intersection of $\mathcal{S}_x$ and $\lambda\mathcal{S}_{x'}$, where λ is a scalar factor equal to the ratio of the row scalar factors of x and x'. Let $(e_1, \ldots, e_k)$ be a basis of $\ker(\Gamma_x^{(i-1)})$, and let $(f_1, \ldots, f_r)$ be a basis of $\ker(\Gamma_{x'}^{(i-1)})$. We then solve the following system in $\mathbb{R}^{d_{i-1}}$:

$$L_x + \mu_1 e_1 + \cdots + \mu_k e_k = \lambda L_{x'} + \lambda_1 f_1 + \cdots + \lambda_r f_r,$$

where the unknowns are $\mu_1, \ldots, \mu_k, \lambda, \lambda_1, \ldots, \lambda_r \in \mathbb{R}$. This system consists of d_{i-1} equations in $\mathbb{R}$. However, only the equations corresponding to the active neurons in the layer $i-1$ for both inputs x and x' are relevant. To ensure that we do not merge spaces from critical points associated with different rows, we overdetermine the system. Indeed, an overdetermined system with random coefficients is inconsistent with very high probability, and thus a system of $\ell > N$ equations with N unknowns typically doesn't have a solution except if there is some ground truth behind it. Therefore, we impose the following merging condition:

$$\ell > 1 + |\ker(\Gamma_x^{(i-1)})| + |\ker(\Gamma_{x'}^{(i-1)})| = 1 + k + r,$$

where ℓ is the number of relevant equations.

Intersections allows us to exploit at least 90% of critical points on the target layer when attacking the

$784 - 8^{(8)} - 1$ MNIST model. As a comparison, the original attack would have solved a system with adequate rank and extracted a correct signature on only 6% of the critical points of layer 4. We could intersect subspaces three by three to have a lower condition for merging, allowing for the use of more critical points. However, merging three by three instead of two by two slows down considerably the attack. For this reason, all our results use two-by-two merges.

As shown in Figure 8, after intersecting critical points with insufficient ranks, the top 7 components yield correct signatures that correspond to neurons in the target layer. Then, after applying our additional discrimination techniques described in Section 4.1 to discard deeper components, all remaining components are in the target layer, as shown in Figure 9. The only unrecovered component corresponds to neuron $\eta_5^{(4)}$, which is always-off (see Figure 10).

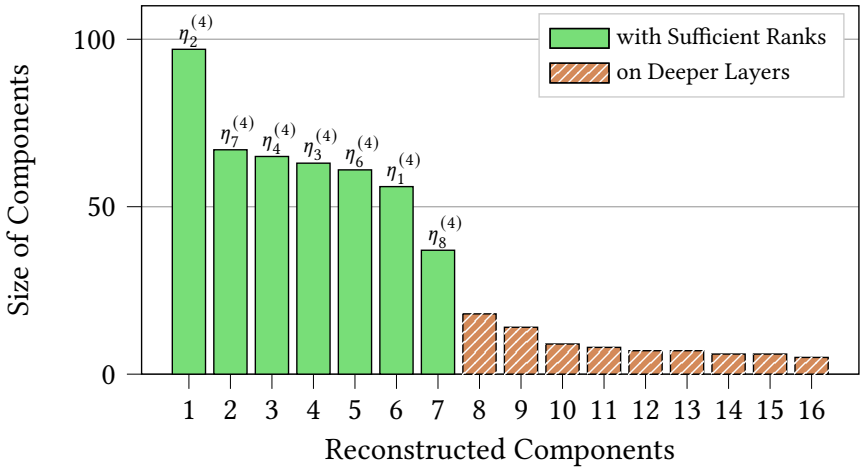

Fig. 8: Merging results after signature intersections. Components with rank issues are discarded, and correct components become prominent. Neuron $\eta_5^{(4)}$ is always off (see Figure 10).

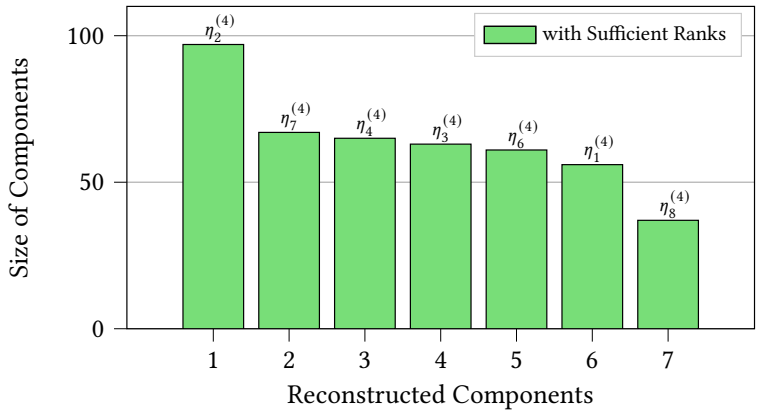

Fig. 9: Final merging results after discarding deeper components. All remaining components are components on the target layer without rank issues. Neuron $\eta_5^{(4)}$ is always off (see Figure 10).

5 Experiments

All our experiments were conducted on an Intel Core i7-14700F CPU using networks trained on either the MNIST or CIFAR-10 datasets, two widely used benchmarking datasets in computer vision. They have also been commonly used in related works [8,6,13] to evaluate model extraction methods. MNIST consists of 28×28 pixel grayscale images of handwritten digits, divided into ten classes ("0" through "9") [17]. In contrast, CIFAR-10 contains 32×32 pixel RGB images of real-world objects across ten classes (e.g., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) [16].

We evaluate our attack on three different networks: one trained on CIFAR-10 with architecture 3072-$8^{(8)}$-1 and two networks trained on MNIST with architectures 784-$8^{(8)}$-1 and 784-$16^{(8)}$-1. The main results are given in Table 1. We refer to those models as Model I, II, and III, respectively. Following [13], we extract the network layer by layer, each time assuming that the previous layers have been correctly recovered. This crucial assumption allows us to evaluate the effectiveness of our improvements, setting aside the floating-point computation imprecisions. State-of-the-art extraction from [13] could not extract any hidden layer past the third layer on a $784 - 16^{(8)} - 1$ MNIST trained network. When attacking the same architecture, we can recover over 95% of the weights in each deeper layer. We apply the attack to other network architectures, yielding similar results. Our improvements largely outperform existing signature extraction methods (Table 1).

We attempted to implement end-to-end model extraction without assuming perfect recovery of the preceding layers, instead relying on previously recovered results to extract subsequent layers. However, for Model II, accumulated floating-point imprecision errors emerged by Layer 5, resulting in incorrect partial signatures. For the same reason, the extraction of Models I and III stopped at layer 3. We also tried to attack a much deeper network with architecture $784 - 8^{(16)} - 1$. The algorithm to discard critical points becomes less effective. This makes deeper components difficult to distinguish based on either τ or size, and intersection computation times grow significantly. Furthermore, because our networks only have width 8 for runtime purposes, when we reach the final four layers (layers 13–16), the solution spaces of critical points almost always have rank 1. The merging condition is rarely met, and we recover no components. We believe this should not happen when attacking wider networks. Therefore we focused on networks with eight hidden layers as those are the deepest considered in the literature of both signature [13] and sign recovery [6].

5.1 Evaluation Metrics

We begin by briefly introducing the metrics used to evaluate the effectiveness of our improved signature extraction:

- **Weight Recovery Rate** denotes the proportion of weights successfully recovered in a layer, including those that can be safely set to zero without impacting the extraction, relative to the total number of weights in that layer. We refer to the other weights as missed weights.
- **Layer Coverage** refers to the percentage of the input space where no unrecovered weight or component of the layer is active.
- **Model Coverage** refers to the percentage of the input space where no unrecovered weight or component of the whole model is active.

Not all weights contribute equally to the network's behaviour, and missing a single weight out of a hundred during extraction does not necessarily imply that the extracted model will behave correctly on 99% of the input space. While [8] introduced the notion of (ϵ, δ)-functional equivalence to capture such nuances, we propose a complementary metric—*coverage*—designed to quantify the proportion of the model that was not accurately recovered. To compute this metric, we randomly sample 10^6 random input points and count how many activate the missed components in the target layer, or activate both a neuron in the previous layer and a neuron in the target layer connected by a missed weight. These input points, where the extracted model diverges from the target due to incomplete recovery, define the *unrecovered space*. The remaining inputs define the *recovered space*. The ratio of the recovered space to the total sampled input space gives the *coverage* score. When evaluating *layer coverage*, the coverage is measured with respect to a single layer. For *model coverage*, coverage is measured across the entire model.

Coverage serves two key purposes. First, it enables us to make a clear distinction between errors coming from imprecisions and errors coming from a lack of information, as it does not depend on the precision of the extraction. Second and foremost, coverage aims to capture the fact that for deep neural networks, we might not recover all the weights and yet still make an extraction correct on most of the input space. If we were to make an (ϵ, δ) assessment of such an extraction, the results would be meaningless except for a δ corresponding to the coverage. Indeed, we would see a sudden increase in ϵ as soon as δ becomes greater than the coverage. (ϵ, δ) assessments are still valuable on the recovered space. Our extraction achieves at least $(0.05, 10^{-8})$ on the recovered part of each network (recall that unlike [8] we normalise ϵ in Definition 6). This means that if we recover each layer in this per-layer setting and recover the signs of the neurons, the output of the extracted network in the recovered space is within 5% of the target network's output. We used 10^8 random points in the recovered space to compute our (ϵ, δ) values.

Table 1: Results demonstrating the extraction of layers deeper than those recovered by state-of-the-art methods in [13]. Each layer is attacked independently using $3,000$ critical points. Layer 9 is linear and can therefore be trivially recovered using previously issued queries.

Model	Signature Extraction	L1*	L2	L3	L4	L5	L6	L7	L8	L9	Entire Model
Ours											
CIFAR-10 3072-$8^{(8)}$-1 (I)	Layer coverage	100%	100%	98.99%	99.45%	97.93%	95.02%	97.09%	95.93%	100%	91.78%
	Weights Recovered	100%	100%	90.63%	98.44%	93.75%	92.19%	95.31%	93.75%	100%	99.91%
	Time	56m48s	1h33m	1h42m	4h43m	4h25m	5h2m	6h53m	2h23m	0.01s	-
	Queries	$2^{23.55}$	$2^{26.15}$	$2^{26.15}$	$2^{27.10}$	$2^{27.04}$	$2^{27.17}$	$2^{27.56}$	$2^{26.45}$	0	-
MNIST 784-$8^{(8)}$-1 (II)	Layer coverage	100%	100%	100%	100%	93.73%	93.73%	76.32%	88.58%	100%	74.12%
	Weights Recovered	100%	100%	100%	100%	90.63%	93.75%	90.63%	95.31%	100%	99.72%
	Time	3m24s	7m5s	7m37s	10m41s	11m46s	28m29s	20m38s	36m46s	0.01s	-
	Queries	$2^{21.59}$	$2^{24.21}$	$2^{24.22}$	$2^{24.24}$	$2^{24.26}$	$2^{25.37}$	$2^{24.98}$	$2^{25.63}$	0	-
MNIST 784-$16^{(8)}$-1 (III)	Layer coverage	100%	100%	100%	93.32%	99.42%	98.66%	90.53%	89.28%	100%	71.35%
	Weights Recovered	100%	100%	100%	99.61%	99.22%	97.27%	98.83%	95.31%	100%	99.83%
	Time	6m17s	26m13s	28m57s	39m33s	1h39m	2h13m	1h48m	3h56m	0.01s	-
	Queries	$2^{21.60}$	$2^{24.22}$	$2^{24.24}$	$2^{24.67}$	$2^{25.65}$	$2^{26.04}$	$2^{25.79}$	$2^{26.95}$	0	-
Foerster et al. [13]											
MNIST 784-$16^{(8)}$-1	Layer coverage	100%	100%	-	-	-	-	-	-	100%	-
	Weights Recovered	100%	100%	-	$\geq 37.50\%^{\dagger}$	-	-	-	$\geq 0\%^{\dagger}$	100%	-
	Time	2h46m	7m50s	-	>36 hours	-	-	-	>36 hours	0.01s	-
	Queries	$2^{22.36}$	$2^{19.01}$	-	-	-	-	-	-	0	-

*'L' stands for layer. **L1** is attacked with 500 critical points. †Foerster et al. recover 6/16 neurons on layer 4 and 0/16 neurons on layer 8.

5.2 Taxonomy of Unrecovered Weights

How does the weight recovery rate relate to coverage? To answer this question we need to understand more about the weights the attack does not recover. For example, as explained earlier (see Section 3.3), some weights are dead weights, meaning they can be set to zero without affecting the coverage. In our attack, we set by default all unrecovered weights to zero, hoping they fall into this category. Clearly it is not the case or the coverage would always sit at 100%. To properly evaluate the quality of the extraction, we must determine whether an unrecovered weight is truly dead. We also aim to understand why some weights are more difficult to recover than others. To this end, we introduce a taxonomy of unrecovered weights that serves both purposes. We classify them into four categories: always-off weights, unreachable inactive weights, unreachable active weights, and query-intensive weights. The first two will turn out to be dead weights. As a preliminary step, Figure 10 illustrates this classification. We then introduce in Section 5.3 a more rigorous set of tests to classify unrecovered weights.

Always-off weights. Always-off neurons are clearly visible in the heat map (Figure 10): neurons $\eta_8^{(2)}$, $\eta_5^{(4)}$, $\eta_2^{(5)}$, $\eta_8^{(5)}$, $\eta_4^{(6)}$, $\eta_3^{(8)}$, and $\eta_6^{(8)}$ are never activated by any input. As a result, all their associated weights are considered always-off, as well as the weights of neurons in the subsequent layer that connect

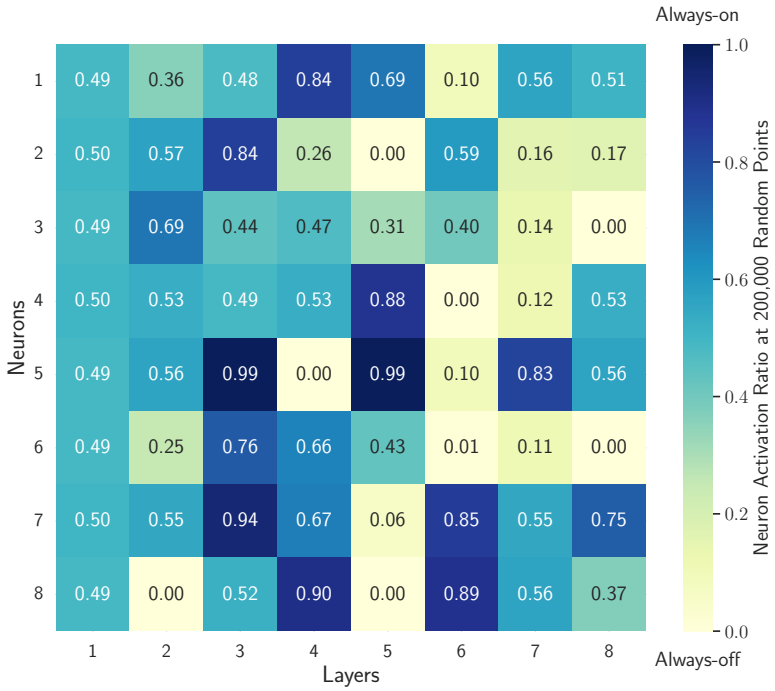

Fig. 10: Neuron activation in Model II based on $200,000$ random input points.

to them. We can safely classify all of these as dead weights. Similarly, a neuron that is always-on, with an activation of 1.00, can also be considered dead (see Section 3.3).

Query-intensive weights. The unrecovered space caused by neurons that are almost always-off or almost always-on (respectively meaning that they are inactive/active on almost all the input space) can be reduced by increasing the number of queries. Making more queries allows us to gather more critical points for these neurons, filling in the unrecovered weights that are hard to find. We refer to them as *query-intensive* weights. However, it is hard to estimate the number of additional queries we need. The number of critical points of a neuron we gather is not clearly related to its activation. This phenomenon is illustrated in Table 2. Sometimes there is a linear relation (layer 2, 6, 8), and sometimes there is no linear relation (layer 4, 7). The correlation indicated is between a layer's neuron's $\min(\text{input activation}, 1 - \text{input activation})$, where input activation is the neuron's number in Table 10, and the ratio of the neuron's critical points over the critical points of the layer. The former represents how close the neuron is to being always-off or always-on. The closer this number is to 0.5, the more its extraction is important. The latter shows how hard it is to find critical points for that neuron (a lower ratio denotes higher difficulty in finding critical points for the neuron). To give a concrete example, $\eta_5^{(7)}$ and $\eta_2^{(7)}$ in Model II have almost the same activation, but we are roughly 2.4 times more likely to find critical points for the latter than for the former. For this reason, it is hard to predict how much time the attack needs to recover more query-intensive weights. We chose $3,000$ critical points as a middle ground between runtime and coverage.

Table 2: Correlation between the value $\min$(activation, 1-activation) of a Model II's neuron (how close the neuron is to being always-off or always-on) and the ratio of the number of its critical points gathered over the total number of critical points on the layer (how hard it is to find critical points for that neuron).

Layer	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Correlation	0.95	0.83	0.58	0.81	0.90	0.65	0.93

Unreachable weights. Suppose that all critical points of a neuron $\eta_j^{(i)}$ lie on the inactive side of a neuron $\eta_k^{(i-1)}$. In this case, the contribution of $\eta_k^{(i-1)}$ to the output of $\eta_j^{(i)}$ is always zero during extraction, leaving us with no information about the corresponding weight $a_{j,k}^{(i)}$. As this weight cannot be recovered using the current extraction method, we refer to such weights as *unreachable*. Unreachable weights arise when a hyperplane from layer i does not intersect with a hyperplane from layer $i-1$, partitioning the input space into three polytopes instead of the usual four. There are two scenarios depending on the sign of $\eta_j^{(i)}$. If no region exists where both neurons are active (as illustrated in Figure 11), then $a_{j,k}^{(i)}$ is never activated and can be safely treated as a dead weight. However, if a region does exist where both neurons are active (depicted in grey in Figure 12), then $a_{j,k}^{(i)}$ does contribute and should ideally be recovered. Unlike query-intensive weights, unreachable weights can significantly reduce extraction coverage, as they are not necessarily associated with almost always-off neurons. Nonetheless, almost always-off neurons are more likely to cause unreachable weights (see Table 4). Fortunately, such cases become increasingly rare with higher input dimensions, since the probability that two (d_0-1)-dimensional hyperplanes do not intersect decreases. For example, a very active unreachable weight, $a_{5,5}^{(7)}$ in Model II, causes a drop in layer coverage to 76.32%, while all other layers maintained values above 88%.

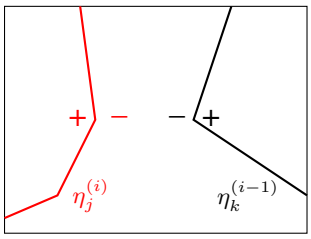

Fig. 11: $a_{j,k}^{(i)}$ is inactive.

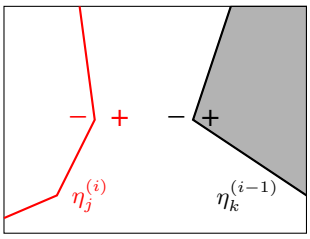

Fig. 12: $a_{j,k}^{(i)}$ is active.

5.3 Application to Our Extractions

While helpful, the heat map is not enough to assess the extraction. For example, it cannot tell us about unreachable weights. We use a combination of three activation tests: input activation, "plus-plus" activation, and "plus-minus" activation. Table 3 indicates how those tests yield the taxonomy. We then apply this classification to analyse our extractions.

Input activation. Input activation corresponds to the heat map from Figure 10. We take $200,000$ random input points, and we check for always-off neurons. Since this test is about neurons and not weights, we also need to set the input activation of weights connected to always-off neurons to 0. This test makes a distinction between always-off weights and unreachable inactive weights.

Plus-plus activation. A weight $a_{j,k}^{(i)}$ represents the contribution of $\eta_k^{(i-1)}$ to $\eta_j^{(i)}$. $a_{j,k}^{(i)}$ contributes to the network when both neurons are active. The plus-plus activation test uses $200,000$ input points to compute the proportion that activates both neurons. This test distinguishes between dead weights and missed weights, and is therefore used to compute coverage.

Plus-minus activation. If $a_{j,k}^{(i)}$ is an active unreachable weight, then there exists no polytope where $\eta_j^{(i)}$ is inactive while $\eta_k^{(i-1)}$ is active. To test for the existence of such a polytope, we again use $200,000$ random input points. Weights that are always off due to an always-off neuron in the previous layer exhibit a plus-minus activation of 0.

We apply this classification to our extractions, showing the results in Table 4. We find that the large majority of unrecovered weights are dead weights: 81%, 83% and 84% for Model I, II and III respectively. This shows that metrics such as the weight recovery rate or the number of neurons fully

Table 3: Taxonomy of unrecovered weights based on the three tests (Input, (+,+) and (+,-) activations) with some examples.

Taxonomy		**Conditions**		
		Input	$(+,+)$	$(+,-)$
Dead Weights	always-off weights	0	0	≥ 0
	unreachable inactive weights	> 0	0	> 0
Missed Weights	unreachable active weights	> 0	> 0	0
	query-intensive weights	> 0	> 0	> 0
Examples				
Model II	$a^{(7)}_{4,5}$	0	0	0
	$a^{(7)}_{3,6}$	0.11	0	0.40
	$a^{(7)}_{5,5}$	0.83	0.11	0
	$a^{(7)}_{2,6}$	0.11	0.11	0.48

recovered are not appropriate. Our taxonomy gives insights into coverage values and what we might expect to see if we increase the number of queries. The high coverage of Model I can be explained by the absence of any major missed weights. More queries should help the extraction of Model III reach a high coverage. However, the presence of three major unreachable weights indicates that more queries might not help tremendously the extraction of Model II. To test this we ran the extraction with twice the amount of critical points and observed that indeed Model II's fidelity barely changed from 74.12% to 74.78% while Model III's changed more substantially from 71.35% to 79.53%, even though for both models we recovered 0.9% points more weights (see Appendix E for details).

Table 4: Proportion and number of each type of unrecovered weights in all extracted models.

Model	**Uncovered Weights**	$(+,+)$	**Number**	**Percentage**
I	always-off weights	0	72	59.50%
	unreachable inactive weights	0	26	21.49%
	unreachable active weights	≤ 0.06	5	4.13%
		> 0.06	0	0%
	query-intensive weights	≤ 0.06	18	14.88%
		> 0.06	0	0%
II	always-off weights	0	84	73.04%
	unreachable inactive weights	0	12	10.43%
	unreachable active weights	≤ 0.06	5	4.35%
		> 0.06	3	2.61%
	query-intensive weights	≤ 0.06	7	6.09%
		> 0.06	4	3.48%
III	always-off weights	0	80	51.61%
	unreachable inactive weights	0	50	32.26%
	unreachable active weights	≤ 0.06	0	0%
		> 0.06	0	0%
	query-intensive weights	≤ 0.06	22	14.19%
		> 0.06	3	1.94%

6 Conclusion

In this work, we carried out an in-depth analysis of the approach introduced in [8] for extracting the weights of neural networks, which has since been reused in several follow-up works. We identified critical limitations that prevented this method from successfully recovering the weights of layers beyond the

third. Most importantly, we proposed algorithmic improvements for each of these challenges, permitting, for the first time, the extraction of over 99% of the weights of 8 hidden layers networks when each layer is attacked independently. For a $3072 - 8^{(8)} - 1$ network, our extracted network's outputs are within 5% of the target network's for more than 91% of possible inputs.

Our results establish a new benchmark for signature extraction and open the way to practical attacks on deeper architectures.

Limitations. Going significantly deeper than eight hidden layers is still a challenge, as are floating-point imprecisions in the context of an end-to-end attack. Addressing these limitations, along with a more thorough investigation of how changes in the architecture affect extraction quality, are promising future areas of research.

References

1. Google cloud platform. Website, https://cloud.google.com/, accessed February 12, 2025
2. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
3. Batina, L., Bhasin, S., Jap, D., Picek, S.: {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 515–532 (2019)
4. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 2–21. Springer, Berlin, Heidelberg (Aug 1991). https://doi.org/10.1007/3-540-38424-3_1
5. Bishop: Bishop Book. https://www.bishopbook.com/ (2025), accessed February 12, 2025
6. Canales-Martínez, I.A., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Satpute, N., Shamir, A.: Polynomial time cryptanalytic extraction of neural network models. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part III. LNCS, vol. 14653, pp. 3–33. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58734-4_1
7. Carlini, N., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Shamir, A.: Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. Cryptology ePrint Archive, Paper 2024/1580 (2024), https://eprint.iacr.org/2024/1580, to be published at EUROCRYPT 2025
8. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 189–218. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_7
9. Chen, Y., Dong, X., Guo, J., Shen, Y., Wang, A., Wang, X.: Hard-label cryptanalytic extraction of neural network models. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VIII. LNCS, vol. 15491, pp. 207–236. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0944-4_7
10. Chen, Y., Dong, X., Guo, J., Shen, Y., Wang, A., Wang, X.: Hard-label cryptanalytic extraction of neural network models. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 207–236. Springer (2024)
11. Daniely, A., Granot, E.: An exact poly-time membership-queries algorithm for extracting a three-layer relu network. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), https://openreview.net/forum?id=-CoNloheTs
12. Fefferman, C., et al.: Reconstructing a neural net from its output. Revista Matemática Iberoamericana **10**(3), 507–556 (1994)
13. Foerster, H., Mullins, R., Shumailov, I., Hayes, J.: Beyond slow signs in high-fidelity model extraction. Neurips **abs/2406.10011** (2024), https://api.semanticscholar.org/CorpusID:270521873
14. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) USENIX Security 2020. pp. 1345–1362. USENIX Association (Aug 2020)
15. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: 29th USENIX security symposium (USENIX Security 20). pp. 1345–1362 (2020)
16. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 and cifar-100 datasets. URl: https://www. cs. toronto. edu/kriz/cifar. html **6**(1), 1 (2009)
17. LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L.: Handwritten digit recognition with a back-propagation network. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems. vol. 2. Morgan-Kaufmann (1989)
18. Lowd, D., Meek, C.: Adversarial learning. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 641–647 (2005)

19. Martinelli, F., Simsek, B., Gerstner, W., Brea, J.: Expand-and-cluster: Parameter recovery of neural networks. In: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net (2024), https://openreview.net/forum?id=3MIuPRJYwf
20. Milli, S., Schmidt, L., Dragan, A.D., Hardt, M.: Model reconstruction from model explanations. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. pp. 1–9 (2019)
21. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning (2010), https://api.semanticscholar.org/CorpusID:15539264
22. Oliynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: A survey on stealing machine learning models and defenses. ACM Computing Surveys **55**(14s), 1–41 (2023)
23. Papernot, N., Mcdaniel, P., Goodfellow, I.J., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (2016), https://api.semanticscholar.org/CorpusID:1090603
24. Reith, R.N., Schneider, T., Tkachenko, O.: Efficiently stealing your machine learning models. In: Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society. pp. 198–210 (2019)
25. Team, H.F.: Hugging face (nd), https://huggingface.co/, accessed February 12, 2025
26. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 601–618. USENIX Association (Aug 2016)
27. Xiao, T., Zhu, J.: Foundations of large language models (2025), https://api.semanticscholar.org/CorpusID:275570622

Appendix

A Example of a Small Network

We illustrate here the definitions introduced in Section 3.1. Consider the $2-3-2-1$ neural network given by the following layers $A^{(i)}, b^{(i)}$:

$$A^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 0.5 & 2 \end{pmatrix}, b^{(1)} = \begin{pmatrix} 1 \\ -2 \\ -5 \end{pmatrix}$$

$$A^{(2)} = \begin{pmatrix} 2 & -4 & -1 \\ -3 & 4 & 5 \end{pmatrix}, b^{(2)} = \begin{pmatrix} -2 \\ -3 \end{pmatrix}$$

$$A^{(3)} = \begin{pmatrix} 1 & -1 \end{pmatrix}, b^{(3)} = \begin{pmatrix} 3 \end{pmatrix}$$

The architecture of this neural network is depicted in Figure 13. Let the input to the neural network be $v = (x, y)$. The neurons in the first layer output $(\eta_1^{(1)}(v), \eta_1^{(2)}(v), \eta_1^{(3)}(v)) = (x+y+1, x-y-2, 0.5x+2y-5)$. Depending on (x, y), the ReLU activation function σ might set some entries to 0 before passing them to the second layer. For example, when the input is $v = (2, 2)$, we have: $F^{(1)}(v) = (5, -2, 0)$, $\sigma \circ F^{(1)}(v) = (5, 0, 0)$, $F^{(2)}(v) = 5 * (2, -3) + 0 * (-4, 4) + 0 * (-1, 5) + (-2, -3) = (8, -18)$, $\sigma \circ F^{(2)}(v) = (8, 0)$, and finally $f(v) = F^{(3)}(v) = 8*1+0*(-1)+3 = 11$. Since $F^{(1)}(v) = (5, -2, 0)$, it follows that v is a critical point of $\eta_3^{(1)}$.

This network can also be represented as a collection of polytopes (see Figure 13). As can be seen in this figure, critical hyperplanes from the first layer make three straight lines, represented as dotted grey lines: $\eta_1^{(1)} : x + y + 1$, $\eta_2^{(1)} : x - y - 2$, and $\eta_3^{(1)} : 0.5x + 2y - 5$. The hyperplanes of $\eta_1^{(2)}$ and $\eta_2^{(2)}$ are shown in their respective colors, with their active side marked by $+$ and their inactive side by $-$. They indeed break on the grey lines as critical hyperplanes on deeper layers break on all hyperplanes of previous layers: the input to $\eta_k^{(i)}$ is $f^{(i-1)}(v)$ rather than v for $i = 2, \cdots, r$.

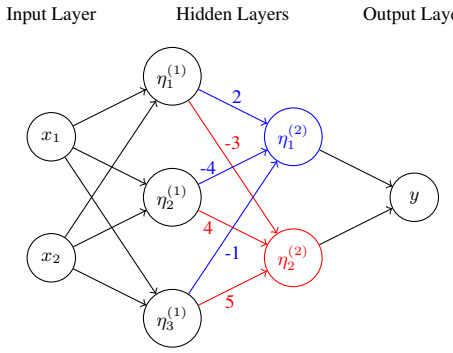

Fig. 13: Network representation of the neural network from Appendix A, with the details of layer 2 highlighted in colour.

Let's compute f_v for $v = (2, 2)$ as an example:

$$\Gamma_v = A^{(3)} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} A^{(2)} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} A^{(1)} = (2.5, 4)$$

$$\gamma_v = f(v) - \Gamma_v \cdot v = 11 - 13 = 2.$$

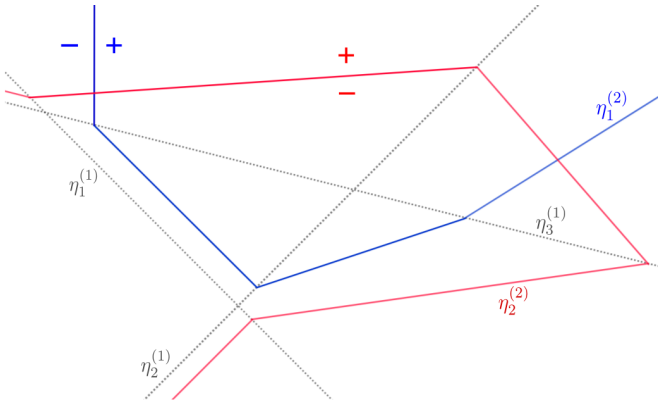

Fig. 14: The neural network from Appendix A partitions the input space into 18 distinct polytopes.

B Finding Critical Points

The attack starts by searching for critical points of neurons on the layer we are targeting. Critical points are at the turning point of a ReLU and thus their left and right derivatives do not agree. We'll exploit this non-linearity to find critical points. We now describe the procedure developed in [8] and in [6]. Very intuitive illustrations are given in both papers.

Suppose we perform the search between two points a and b on the search line. First we compute the derivatives of f at a towards b and of f at b towards a, respectively m_a and m_b. If they are the same then we know that there are no critical points on the search line. If they are not the same we know that there is at least a critical point on the search line between a and b. We could directly reiterate the binary search on each half of the segment, etc. There is however a more efficient way to find the critical points. It

relies on predicting where the critical point should be if it is the only one between a and b, and checking if it is indeed there.

If there is a unique critical point, we expect to find it where the derivatives cross at $x^* = a + \frac{f(b)-f(a)-m_b(b-a)}{a-b}$. Its expected value is $\hat{f}(x^*) = f(a) + m_a \frac{f(b)-f(a)-m_b(b-a)}{a-b}$. If $\hat{f}(x^*) = f(x)$ then the critical point is unique, otherwise we keep dividing the segment with dichotomy. This faster algorithm can generate pathological errors, see ([6], p.32) for more conditions to deal with them. In practice, we use [8]'s strategy.

C Details of Signature Extraction from [8]

C.1 Proof of Lemma 2

Let x be a critical point of layer $i \in \{1, \dots, r\}$ for neuron $\eta_k^{(i)}$ such that x is not a critical point of any other neuron, and $\Delta \in \mathbb{R}^{d_0}$. Then, denoting $\Omega_x^{(i+1)} = (c_1, \dots, c_{d_i})$, we have:

$$\partial_\Delta^2 f(x) = c_k \left| \Gamma_x^{(i-1)} \Delta \cdot A_k^{(i)} \right|$$

Proof. We take ϵ sufficiently small so that $x + \epsilon\Delta$ and $x - \epsilon\Delta$ are in the neighbourhood of x. Thus $x + \epsilon\Delta$ and $x - \epsilon\Delta$ have the exact same activation pattern except for $\eta_k^{(i)}$.

$$\begin{aligned}
\partial_\Delta^2 f(x) &= \frac{1}{\epsilon}(f(x+\epsilon\Delta) + f(x-\epsilon\Delta) - 2f(x)) \\
&= \frac{1}{\epsilon}\Big(G^{(i+1)} \circ \sigma(A^{(i)} F^{(i-1)}(x+\epsilon\Delta) + b^{(i)}) \\
&+ G^{(i+1)} \circ \sigma(A^{(i)} F^{(i-1)}(x-\epsilon\Delta) + b^{(i)}) \\
&- 2\, G^{(i+1)} \circ \sigma(A^{(i)} F^{(i-1)}(x) + b^{(i)}) \Big) \\
&= \frac{1}{\epsilon} \Omega_x^{(i+1)} \Big(\sigma(A^{(i)} F^{(i-1)}(x+\epsilon\Delta) + b^{(i)}) \\
&+ \sigma(A^{(i)} F^{(i-1)}(x-\epsilon\Delta) + b^{(i)}) \\
&- 2\, \sigma(A^{(i)} F^{(i-1)}(x) + b^{(i)}) \Big)
\end{aligned}$$

Let us analyze $C = \sigma(A^{(i)} F_x^{(i-1)}(x+\epsilon\Delta)+b^{(i)}) + \sigma(A^{(i)} F_x^{(i-1)}(x-\epsilon\Delta)+b^{(i)}) - 2\sigma(A^{(i)} F_x^{(i-1)}(x)+b^{(i)}) \in \mathbb{R}^{d_i \times 1}$.

The coefficients $j \neq k$ of C are zero. Indeed, since x, $x + \epsilon\Delta$, and $x - \epsilon\Delta$ belong to the same activation region except for $\eta_k^{(i)}$, the computation becomes affine, leading to zero. More precisely:

$$\begin{aligned}
C[j] \;=\; & [\Gamma_x^{(i-1)}(x+\epsilon\Delta) + \gamma_x^{(i-1)}] \cdot A_j^{(i)} \\
& + [\Gamma_x^{(i-1)}(x-\epsilon\Delta) + \gamma_x^{(i-1)}] \cdot A_j^{(i)} \\
& - 2[\Gamma_x^{(i-1)} x + \gamma_x^{(i-1)}] \cdot A_j^{(i)} \\
& = \Gamma_x^{(i-1)} x \cdot A_j^{(i)} + \epsilon \Gamma_x^{(i-1)} \Delta \cdot A_j^{(i)} \\
& + \; \Gamma_x^{(i-1)} x \cdot A_j^{(i)} \; - \; \epsilon \Gamma_x^{(i-1)} \Delta \cdot A_j^{(i)} - 2\Gamma_x^{(i-1)} x \cdot A_j^{(i)} \\
& = 0 \quad \text{when } \eta_j^{(i)} \text{ is active at } x
\end{aligned}$$

$C[j] = 0 + 0 - 2 \times 0 = 0$ when $\eta_j^{(i)}$ is inactive at x.

For coefficient k, since x is a critical point of $\eta_k^{(i)}$, we have:

$$F^{(i-1)}(x) \cdot A_k^{(i)} \; + \; b_k^{(i)} = [\Gamma_x^{(i-1)} x + \gamma_x^{(i-1)}] \cdot A_k^{(i)} \; + \; b_k^{(i)} = 0$$

and since $\Gamma_x^{(i-1)}\Delta \cdot A_k^{(i)}$ is either positive or negative, we get:

$$C[k] = \left|\Gamma_x^{(i-1)}\Delta \cdot A_k^{(i)}\right|$$

Thus, by multiplying $\Omega_x^{(i+1)}$ by C, we obtain the expected result.

C.2 Correlating the Weights' Signs

We know by Lemma 2 that

$$\frac{\partial^2_{\Delta} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{\left|(\Gamma_x^{(i-1)}\Delta) \cdot A_k^{(i)}\right|}{\left|(\Gamma_x^{(i-1)}\Delta_0) \cdot A_k^{(i)}\right|} \coloneqq \frac{|\alpha|}{|\beta|}$$

Now consider on one hand,

$$\frac{\partial^2_{\Delta+\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{\left|\alpha+\beta\right|}{\left|\beta\right|}$$

And on the other,

$$\frac{\partial^2_{\Delta} f(x) + \partial^2_{\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{\left|\alpha\right| + \left|\beta\right|}{\left|\beta\right|}$$

The two values above are equal if and only if α and β have the same sign. This additional test allows us to remove the absolute values and correlate the sign of our result for each direction Δ we take to that of the first direction Δ_0 taken. This is why the sign recovery only has to find one sign per neuron rather than one sign per weight.

C.3 Solving for the Signature

By querying different directions $\{\Delta_m\}$ around x a critical point of $\eta_k^{(i)}$, we can obtain a set of $\{y_m\}$ such that,

$$y_m = \frac{(\Gamma_x^{(i-1)}\Delta_m) \cdot A_k^{(i)}}{c_k^{(i)}},$$

where $c_k^{(i)} = (\Gamma_x^{(i-1)}\Delta_0) \cdot A_k^{(i)}$. To solve for $A_k^{(i)}$, we build the following system of equations:

$$\begin{pmatrix} \Gamma_x^{(i-1)}\Delta_1 \\ \Gamma_x^{(i-1)}\Delta_2 \\ \vdots \\ \Gamma_x^{(i-1)}\Delta_{d_{i-1}+1} \end{pmatrix} \cdot \frac{1}{c_k^{(i)}} \begin{pmatrix} a_{k,1}^{(i)} \\ a_{k,2}^{(i)} \\ \vdots \\ a_{k,d_{i-1}+1}^{(i)} \end{pmatrix} = \begin{pmatrix} 1 \\ y_2 \\ \vdots \\ y_{d_{i-1}+1} \end{pmatrix}$$

When extracting the first layer, the input of the layer is also the input to the network, giving us full control over it. We can use the input basis vectors $\{e_i\}$ as directions Δ, allowing us to recover each entry of the signature independently from each equation. However, for deeper layers, we must first gather enough linearly independent conditions.

D Deeper Critical Points Merging with Components on the Target Layer

In this section, we show that a critical point x_1 of neuron $\eta_k^{(i)}$ on the target layer can merge with a critical point x_2 of a neuron in a deeper layer $i+t$ if x_2 causes $\eta_k^{(i)}$ to be the only active neuron on its layer, meaning that for some $a_k \in \mathbb{R}$,

$$\sigma \circ F^{(i)}(x_2) = (0, \ldots, 0, a_k, 0, \ldots, 0).$$

We note y_1 and y_2 the matrix rows extracted from x_1 and x_2 respectively. We therefore have,

$$\partial^2_{\Delta} f(x_1) = (\Gamma^{(i-1)}_{x_1} \Delta) \cdot y_1$$
$$\partial^2_{\Delta} f(x_2) = (\Gamma^{(i+t-1)}_{x_2} \Delta) \cdot y_2$$

Let's write A_{x_2} the matrix $A'_{x_2} \circ I^{(i)}_{x_2} \circ \ell^{(i)}$, where $A'_{x_2} = \ell^{(i+t-1)} \circ I^{(i+t-2)}_{x_2} \circ \cdots \circ I^{(i+1)}_{x_2} \circ \ell^{(i+1)}$. Thus, for $c \in \mathbb{R}$,

$$\begin{aligned}
(\Gamma^{(i+t-1)}_{x_2} \Delta) \cdot y_2 &= [(A_{x_2} \circ \Gamma^{(i-1)}_{x_2}) \Delta] \cdot y_2 \\
&= (\Gamma^{(i-1)}_{x_2} \Delta) \cdot (A^{\top}_{x_2} y_2) \\
&= (\Gamma^{(i-1)}_{x_2} \Delta) \cdot [(\ell^{(i)\top} \circ I^{(i)\top}_{x_2} \circ A'^{\top}_{x_2}) y_2] \\
&= (\Gamma^{(i-1)}_{x_2} \Delta) \cdot c y_1
\end{aligned}$$

Let's explain the last equality. Our assumption that x_2 causes $\eta^{(i)}_k$ to be the only active neuron on its layer implies that $I^{(i)}$ is diagonal with all entries zero except for a one at the k-th diagonal position. We also have that y_1 is the k-th row of the matrix $\ell^{(i)}$. Therefore, $(\ell^{(i)\top} \circ I^{(i)\top}_{x_2} \circ A'^{\top}_{x_2}) y_2 = c \cdot y_1$, where c is a constant. The partial signatures y_1 and $c \cdot y_1$ extracted from x_1 and x_2 can be thus merged even though x_2 is on a deeper layer.

E Extraction Results Using 6,000 Critical Points

Table 5: Additional results on the proportion and count of each type of unrecovered weight in models II and III, extracted with $3,000$ or $6,000$ critical points.

Model	**Uncovered Weights**	$(+,+)$	**Number**	**Percentage**
II	always-off weights	0	84	73.04%
$3,000$ CPs	unreachable inactive weights	0	12	10.43%
	unreachable active weights	≤ 0.06	5	4.35%
		> 0.06	3	2.61%
	query-intensive weights	≤ 0.06	7	6.09%
		> 0.06	4	3.48%
II	always-off weights	0	84	78.50%
$6,000$ CPs	unreachable inactive weights	0	10	9.35%
	unreachable active weights	≤ 0.06	5	4.67%
		> 0.06	3	2.80%
	query-intensive weights	≤ 0.06	1	0.93%
		> 0.06	4	3.74%
III	always-off weights	0	80	51.61%
$3,000$ CPs	unreachable inactive weights	0	50	32.26%
	unreachable active weights	≤ 0.06	0	0%
		> 0.06	0	0%
	query-intensive weights	≤ 0.06	22	14.19%
		> 0.06	3	1.94%
III	always-off weights	0	72	69.90%
$6,000$ CPs	unreachable inactive weights	0	24	23.30%
	unreachable active weights	≤ 0.06	0	0%
		> 0.06	0	0%
	query-intensive weights	≤ 0.06	6	5.83%
		> 0.06	1	0.97%

Table 6: Additional extraction results on models II and III. Each layer is attacked independently with $3,000$ or $6,000$ critical points.

Model	Signature Extraction	L2*	L3	L4	L5	L6	L7	L8	Entire Model
					Ours				
MNIST	Layer coverage	100%	100%	100%	93.73%	93.73%	76.32%	88.58%	74.12%
$784-8^{(8)}-1$ (II)	Weights Recovered	100%	100%	100%	90.63%	93.75%	90.63%	95.31%	99.72%
$3,000$ critical points	Time	7m5s	7m37s	10m41s	11m46s	28m29s	20m38s	36m46s	-
	Queries	$2^{24.21}$	$2^{24.22}$	$2^{24.24}$	$2^{24.26}$	$2^{25.37}$	$2^{24.98}$	$2^{25.63}$	-
MNIST	Layer coverage	100%	100%	100%	99.99%	93.76%	76.32%	88.65%	74.78%
$784-8^{(8)}-1$ (II)	Weights Recovered	100%	100%	100%	98.44%	95.31%	90.63%	95.31%	99.81%
$6,000$ critical points	Time	14m10s	15m57s	28m25s	29m1s	38m41s	38m14s	56m15s	-
	Queries	$2^{25.21}$	$2^{25.22}$	$2^{25.24}$	$2^{25.35}$	$2^{25.83}$	$2^{25.79}$	$2^{26.18}$	-
MNIST	Layer coverage	100%	100%	93.32%	99.42%	98.66%	90.53%	89.28%	71.35%
$784-16^{(8)}-1$ (III)	Weights Recovered	100%	100%	99.61%	99.22%	97.27%	98.83%	95.31%	99.83%
$3,000$ critical points	Time	26m13s	28m57s	39m33s	1h39m	2h13m	1h48m	3h56m	-
	Queries	$2^{24.22}$	$2^{24.24}$	$2^{24.67}$	$2^{25.65}$	$2^{26.04}$	$2^{25.79}$	$2^{26.95}$	-
MNIST	Layer coverage	100%	98.79%	100%	100%	99.73%	91.35%	89.54%	79.53%
$784-16^{(8)}-1$ (III)	Weights Recovered	100%	99.61%	100%	100%	99.61%	99.61%	98.44%	99.91%
$6,000$ critical points	Time	55m3s	58m5s	1h4m	2h8m	2h35m	3h6m	5h1m	-
	Queries	$2^{25.22}$	$2^{25.24}$	$2^{25.27}$	$2^{25.68}$	$2^{25.88}$	$2^{26.43}$	$2^{27.23}$	-

* 'L' stands for layer.

Table 5 shows that as the number of critical points increases, the number of query-intensive weights decreases. This suggests that some of these weights can be activated and recovered, leading to higher model fidelity. The number of dead weights also drops with more critical points, as weight plus-plus activation and neuron activation values below 0.001 are initially considered always-off. With more critical points, these weights and neurons can become active. However, the number of unreachable active weights remains unchanged, continuing to limit full model extraction.

Table 6 presents the extraction results on models II and III using 6,000 critical points, compared to those using 3,000. Doubling the number of critical points generally improves both layer fidelity and weight recovery rate in deeper layers, as more query-intensive weights can be activated and recovered. Two notable exceptions are observed. First, in layer 8 of model II, the weight recovery rate is identical for both settings, but layer fidelity improves. This is because some dead weights for 3,000 critical points became active for 6,000 critical points, and were then recovered. It does not impact the weight recovery rate but slightly improves layer fidelity. Second, in layer 3 of model III, both the weight recovery rate and layer fidelity are lower with 6,000 critical points. This is due to the randomness in critical point selection. A weight with a weight plus-plus activation value of 0.012 is only activated among the 3,000 critical points set.