Is the Hard-Label Cryptanalytic Model Extraction Really Polynomial?

Akira Ito[1], Takayuki Miura[2], and Yosuke Todo[2]

[1] Tohoku University
2–1–1 Katahira, Aoba-ku, Sendai-shi, 980-8577, Japan
akira.ito.b1@tohoku.ac.jp,
[2] NTT Social Informatics Laboratories,
3–9–11 Midori-cho, Musashino-shi, Tokyo, 180-8585, Japan
tkyk.miura@ntt.com, yosuke.todo@ntt.com

Abstract. Deep Neural Networks (DNNs) have attracted significant attention, and their internal models are regarded as valuable intellectual assets. Extracting a model via oracle access to a DNN is conceptually similar to extracting a secret key via oracle access to a block cipher. Consequently, cryptanalytic techniques, particularly differential-like attacks, have been actively explored recently. ReLU-based DNNs are the most common architectures. While early works (e.g., Crypto 2020, Eurocrypt 2024) assume access to exact output logits, more recent works (e.g., Asiacrypt 2024, Eurocrypt 2025) focus on the hard-label setting, where the attacker observes only the final classification result (e.g., "dog" or "car"). Notably, Carlini et al. (Eurocrypt 2025) demonstrated that model extraction is feasible in polynomial time even under this restricted setting. In this paper, we show that a key assumption of their attack becomes increasingly unrealistic as the target depth grows. While prior works have noted neurons whose activation states rarely change, we analyze their concrete impact on hard-label extraction: even a single neuron that is (almost) always active can prevent the attack from proceeding unless its parameters are recovered, and ignoring it inevitably incurs a non-negligible error. A straightforward solution is to recover these parameters from a state switch, but observing such a switch becomes exponentially harder as the depth increases, implying that hard-label extraction is not always polynomial time. To address this limitation, we propose *cross-layer extraction*. Rather than extracting the secret parameters (e.g., weights and biases) directly, we exploit cross-layer interactions to recover them from deeper layers, reducing query complexity and addressing limitations of existing model extraction approaches. The source code is available at https://github.com/ECSIS-lab/hard-label-cross-layer-extraction.

Keywords: ReLU network · Model extraction · Hard-label attack.

1 Introduction

Deep Neural Networks (DNNs) have become one of the most fundamental AI technologies and play an important role in various aspects of our society. Well-

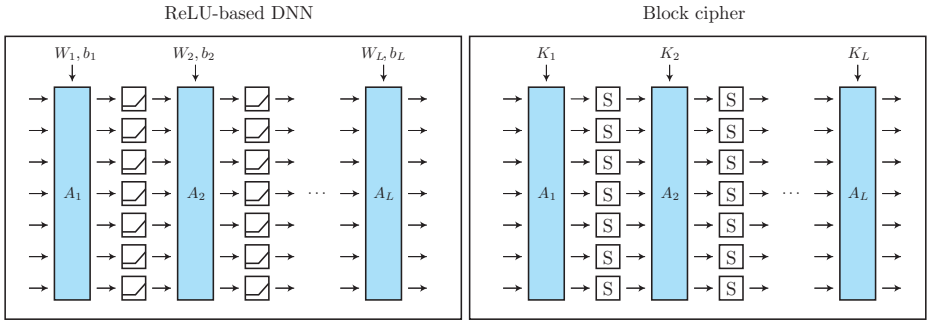


Fig. 1: ReLU-based DNN and block cipher

trained models can perform complex tasks, such as image classification and natural language processing, often surpassing human-level performance [24,20,21]. Therefore, they are regarded as intellectual assets, and their security has become a critical concern. If a (malicious) user is able to extract the learned model, it poses a significant risk to the model provider [16,13,10,9,18,23,22,19].

ReLU-based DNNs are widely deployed due to their simplicity and effectiveness. Evaluating the feasibility of model extraction, in which an attacker extracts model parameters solely from input-output behavior, is therefore particularly important for ReLU-based DNNs. Recently, cryptographers have shown increasing interest in this problem because, from a high-level point of view, ReLU-based DNNs share structural similarities with block ciphers. Consequently, cryptanalytic techniques are a promising approach to model extraction.

Fig. 1 illustrates the structural comparison between ReLU-based DNNs and block ciphers. As the figure shows, DNNs and block ciphers exhibit notable similarities. Both share an iterative structure in which each step consists of a linear (more precisely, affine) layer and a nonlinear layer. In block ciphers, the only nonlinear components are the S-boxes, while in DNNs, the ReLU functions serve as the sole nonlinear elements. Common block ciphers such as AES [8] are constructed from public linear layers and S-boxes, with secret subkeys derived from a key schedule. More general constructions, such as SASAS [2] or ASASA [1], incorporate entirely secret affine and nonlinear layers. In contrast, the affine layers of DNNs are typically not publicly disclosed, and constitute the core intellectual asset of the model. The multiplication matrices in the linear layers are referred to as *weights*, and the added secret vectors are referred to as *biases*.

On the other hand, it would be overly simplistic to assume that DNNs and block ciphers are identical. When considering model extraction attacks, the most crucial difference lies in the piecewise linearity of DNNs. In block ciphers, even a 1-bit (or more generally, an arbitrary) difference significantly alters the nonlinear behavior. To ensure sufficient diffusion, designers guarantee a large number of active S-boxes. In contrast, let f be a ReLU network. For a given input $\boldsymbol{x} \in \mathbb{R}^{d_0}$, the function f is locally affine around $\boldsymbol{x}$. That is, for sufficiently small $\boldsymbol{\delta} \in \mathbb{R}^{d_0}$,

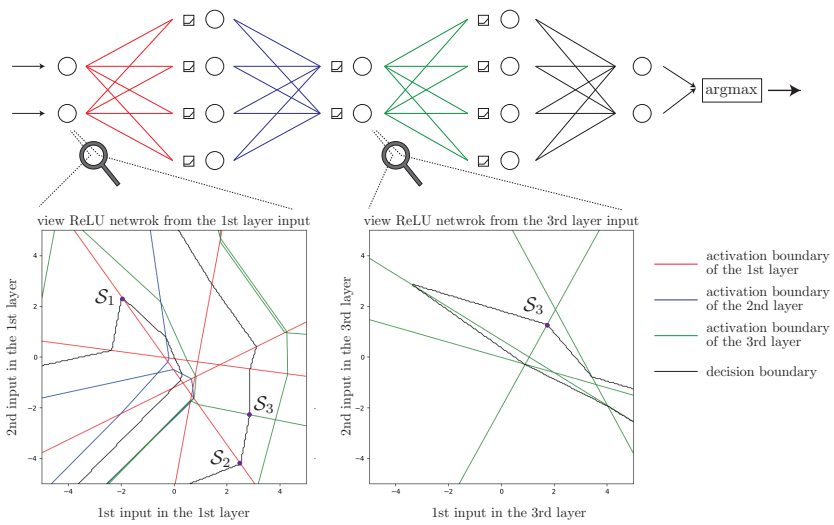


Fig. 2: Overview of model extraction with hard-label setting.

there exists a matrix $\mathbf{F}_{\boldsymbol{x}}$ dependent on $\boldsymbol{x}$, such that

$$f(\boldsymbol{x}+\boldsymbol{\delta}) = \mathbf{F}_{\boldsymbol{x}}\boldsymbol{\delta} + f(\boldsymbol{x}).$$

This local linearity is one of the key reasons why polynomial-time model extraction attacks are feasible for DNNs.

Early works [5,17,7,15] assume that the attacker has access to the exact output logits, which are the direct output of the ReLU network. However, in realistic scenarios, it is unlikely that all logits are exposed to users. For example, in MNIST classification, the network produces a 10-dimensional vector of scores (logits), but the final output is only the predicted class, obtained by taking the argmax over these 10 values. Therefore, more recent works have shifted their focus to the *hard-label setting* [6,4,3], where only the final classification result is visible to the attacker. Chen et al. proposed the first model extraction under this setting, but their method applies only to limited models and requires exponential time complexity [6]. Later, Carlini et al. proposed a new model extraction that is applicable to multilayer perceptrons (MLPs) and runs in polynomial time [4].

Carlini et al.'s attack inserts sufficiently small differences $\boldsymbol{\delta}$ and detects a point, called an *intersection point*, where the network's local linear behavior changes. It also detects an affine subspace, called an *intersection space*.[3] Fig. 2 provides a high-level view of why model extraction is feasible. Owing to the

[3] Carlini et al. used different terms, *dual point* and *dual space*, for *intersection point* and *intersection space*, respectively. Here, "dual" refers to points/space on both activation and decision boundaries, and it differs from the standard notion of a dual space in mathematics. Therefore, to avoid misinterpretation, we use the term intersection point/intersection space instead.

piecewise linearity of a ReLU network, the input space is partitioned into linear regions. Adjacent regions are separated by an *activation boundary*, where exactly one neuron switches between active and inactive. Within each region, the activation pattern of all neurons is fixed, and the *decision boundary* (where the output label changes) behaves linearly. By observing this local linear behavior, an attacker can identify intersection points and, more generally, intersection spaces, where a decision boundary intersects an activation boundary. Note that all intersection spaces are defined in the input space, e.g., $\mathcal{S}_3$, which lies at the intersection between the decision boundary and the activation boundary of the 3rd layer, is visible from the input. Two intersection spaces, $\mathcal{S}_1$ and $\mathcal{S}_2$, can lie on the same first-layer activation boundary. In this case, the attacker can identify the corresponding activation boundary due to its linear nature, thereby recovering the associated secret parameters (weights and biases). In contrast, activation boundaries in deeper layers (shown in blue or green) appear nonlinear when viewed from the input space. However, as illustrated in the figure, when the network is viewed from an intermediate layer (e.g. the 3rd layer), these boundaries become linear. Therefore, once first-layer parameters are recovered, the same strategy can be applied recursively, layer by layer.

1.1 Our Contribution

Is the Hard-Label Extraction Really Polynomial? Carlini et al.'s work is remarkably powerful. They claimed that hard-label model extraction is possible in polynomial time in the number of neurons and the network depth. However, their argument implicitly assumes that intersection points for any target neuron can be collected with a polynomial number of queries. This assumption is reasonable when neuron activation states switch with roughly uniform probability.

Does this assumption hold in practice? As shown in Fig. 2, while intersection points exist between the decision boundary and the activation boundary of the 3rd layer, the ReLU nonlinearity restricts the region that is reachable by the attack. Consequently, if the corresponding intersection points lie outside the region reachable from the input (e.g., outside the all-positive region), they become inaccessible. Even when reachable intersection points exist, finding them can be extremely difficult if the reachable intersection region is very narrow.

To validate our intuition, we conducted experiments using trained networks with high classification accuracy in Section 4.2. We find that inaccessible intersection points correspond to neurons whose activation states remain fixed, either always active or always inactive, under many queries. Moreover, as depth increases, certain neurons require an exponential number of queries to switch their activation state. We refer to these neurons as *persistent* (always active) and *dead* (always inactive), respectively.

Persistent/dead neurons have been noted in prior works [17,4,15], but their implications for subsequent layers in model extraction have not been fully analyzed.[4] In contrast to prior works, we show that the existence of (almost) per-

[4] Canales-Martinez et al. [17] mention the possibility of persistent neurons but report finding none in their model. Carlini et al. [4] remark that such neurons are rare. Liu

sistent neurons constitutes a fundamental challenge to existing hard-label model extraction. When intersection points cannot be found, the weights and biases of these neurons cannot be recovered. While dead neurons are not critical, as their weights and biases do not influence the final output, the weights of persistent neurons directly affect the behavior of subsequent layers. We show that ignoring these weights inevitably causes non-negligible errors in the extracted model, preventing the attack from proceeding correctly.

Consequently, if almost persistent neurons exist, a successful approach requires an exponential number of queries until they become inactive, which is no longer polynomial time. Moreover, when neurons are truly persistent, model extraction becomes impossible, even with an exponential number of queries.

Cross-Layer Extraction: Extracting Persistent Neurons. To address the challenges posed by persistent neurons, we propose a new technique called *cross-layer extraction*. When intersection points cannot be found for (almost) persistent neurons, cross-layer extraction enables their recovery by leveraging interactions across layers. Specifically, it uses intersection points from neurons in the next layer, which are easier to detect. Moreover, this technique can also identify the number of persistent and dead neurons. Cross-layer extraction does not necessarily recover the exact target model. Instead, it reproduces the original model's outputs as long as neurons classified as persistent or dead remain in those respective states. Since the probability of these states changing is exponentially small, the recovered model produces identical outputs with overwhelming probability. Thus, cross-layer extraction reconstructs the best possible model given the information available to the attacker. This stands in contrast to existing approaches, which either fail entirely or incur exponential query cost.

For reproducibility, our implementation is available at `https://github.com/ECSIS-lab/hard-label-cross-layer-extraction`.

2 Preliminaries

In this section, we describe properties of neural networks with ReLU activations. In addition, we present the adversarial capabilities and objectives in the model extraction attacks considered in this paper. The notation used throughout this paper is summarized in Table 1.

Definition 1 (ReLU [11]). *A ReLU function, denoted by* $\sigma : \mathbb{R}^{d_*} \to \mathbb{R}^{d_*}$*, is defined by* $\sigma(\boldsymbol{x})_i = \max\{x_i, 0\}$ *for each* $1 \leq i \leq d_*$*.* [5]

et al. [15] observe almost such neurons but state that there is no need to identify them individually as absorbable for the recovery of the final layer.

[5] Although this definition depends on the input dimension, it is customary to denote it by the same symbol σ, since it applies the same operation component-wise to the input vector.

Table 1: Notation used in this paper

Symbol	Description
$\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$	weight matrix in the ℓth layer
$\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$	bias vector in the ℓth layer
$\boldsymbol{z}^{(\ell)} \in \mathbb{R}^{d_\ell}$	output vector after the ℓth layer
$f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$	ReLU network with L layers
$f^{(\ell)} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_\ell}$	ReLU network up to the ℓth layer
σ	ReLU function
$\mathbf{D}_{\boldsymbol{x}}^{(\ell)}$	diagonal matrix encoding the ℓth-layer activation pattern at $\boldsymbol{x}$
$\boldsymbol{s}$	intersection point
$\mathcal{S}(\boldsymbol{s}, \mathbf{N})$	intersection space
$\mathbf{F}_x^{(\ell)}$	forward local linear matrix up to the ℓth layer for input $\boldsymbol{x}$

Definition 2 (ReLU Network). *Let $L \in \mathbb{Z}_{>0}$ be a positive integer. For $1 \leq \ell \leq L-1$, we define*

$$\boldsymbol{z}^{(\ell)} = \sigma(\mathbf{W}^{(\ell)} \boldsymbol{z}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}),$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$, $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$, and $\boldsymbol{z}^{(0)} = \boldsymbol{x}$. In general, the input and output of the ReLU function are called the pre-activation and the activation, respectively. Here, the pre-activation at layer ℓ is given by $\mathbf{W}^{(\ell)} \boldsymbol{z}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}$, and the activation is denoted by $\boldsymbol{z}^{(\ell)}$. We also define $\boldsymbol{z}^{(L)} = \mathbf{W}^{(L)} \boldsymbol{z}^{(L-1)} + \boldsymbol{b}^{(L)}$, as the last layer, where $\mathbf{W}^{(L)} \in \mathbb{R}^{d_L \times d_{L-1}}$, $\boldsymbol{b}^{(L)} \in \mathbb{R}^{d_L}$. Then, an L-layer **ReLU network**[6] *is a function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ such that $f(\boldsymbol{x}) = \boldsymbol{z}^{(L)}$. In particular, we also denote $f^{(\ell)}(\boldsymbol{x}) = \boldsymbol{z}^{(\ell)}$. A list $\theta = \{(\mathbf{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})\}_{\ell=1,\dots,L}$ is called the parameters of the ReLU network, sometimes also referred to as the weights and biases. The weight of the kth neuron in the ℓth layer, denoted by $\boldsymbol{w}_k^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$, is equal to the transpose of the kth row vector of the weight matrix $\mathbf{W}^{(\ell)}$. Here, we assume that the neural network is a hard-label model. By using a labeling function $\hat{y} : \mathbb{R}^{d_L} \to [d_L]$, the hard-label model is denoted by $\hat{y} \circ f : \mathbb{R}^{d_0} \to [d_L]$.*

When considering a neural network with hard labels, it can be assumed that the weights of all neurons in each layer are normalized to have unit norm. This is because, through an appropriate transformation, one can obtain weights that yield the same outputs for any given input. Furthermore, within each layer, the order of neurons can be rearranged so as to coincide by inserting a suitable permutation matrix in between.

A ReLU network f is locally affine. Namely, for $\boldsymbol{x} \in \mathbb{R}^{d_0}$, there exist a region $R \subset \mathbb{R}^{d_0}$, a matrix $\mathbf{F}_{\boldsymbol{x}}$ and a vector $\boldsymbol{\beta}_{\boldsymbol{x}}$ such that $f(\boldsymbol{y}) = \mathbf{F}_{\boldsymbol{x}} \boldsymbol{y} + \boldsymbol{\beta}_{\boldsymbol{x}}$ for all $\boldsymbol{y} \in R$. Let us be more specific. For each point $\boldsymbol{x} \in \mathbb{R}^{d_0}$, the ReLU network has an activation pattern represented by diagonal matrices $\mathbf{D}_{\boldsymbol{x}}^{(1)}, \dots, \mathbf{D}_{\boldsymbol{x}}^{(L-1)}$. Here, $(\mathbf{D}_{\boldsymbol{x}})_{kk}^{(\ell)} = 1$ indicates that the kth neuron in the ℓth layer is activated and $(\mathbf{D}_{\boldsymbol{x}})_{kk}^{(\ell)} = 0$ means that its value is set to zero by the ReLU. Then, we can

[6] In this paper, when we refer to a ReLU network, we mean a multi-layer perceptron whose activation function is the ReLU.

describe the forward local linear matrix $\mathbf{F}_{\boldsymbol{x}}^{(\ell)}$ and bias $\boldsymbol{\beta}_{\boldsymbol{x}}^{(\ell)}$ explicitly as follows:

$$\mathbf{F}_{\boldsymbol{x}}^{(\ell)} = \mathbf{D}_{\boldsymbol{x}}^{(\ell)}\mathbf{W}^{(\ell)}\mathbf{D}_{\boldsymbol{x}}^{(\ell-1)}\mathbf{W}^{(\ell-1)}\cdots\mathbf{D}_{\boldsymbol{x}}^{(1)}\mathbf{W}^{(1)},$$
$$\boldsymbol{\beta}_{\boldsymbol{x}}^{(\ell)} = \mathbf{D}_{\boldsymbol{x}}^{(\ell)}\boldsymbol{b}^{(\ell)} + \mathbf{D}_{\boldsymbol{x}}^{(\ell)}\mathbf{W}^{(\ell)}\mathbf{D}_{\boldsymbol{x}}^{(\ell-1)}\boldsymbol{b}^{(\ell-1)} + \cdots + \mathbf{D}_{\boldsymbol{x}}^{(\ell)}\mathbf{W}^{(\ell)}\mathbf{D}_{\boldsymbol{x}}^{(\ell-1)}\cdots\mathbf{D}_{\boldsymbol{x}}^{(1)}\boldsymbol{b}^{(1)}.$$

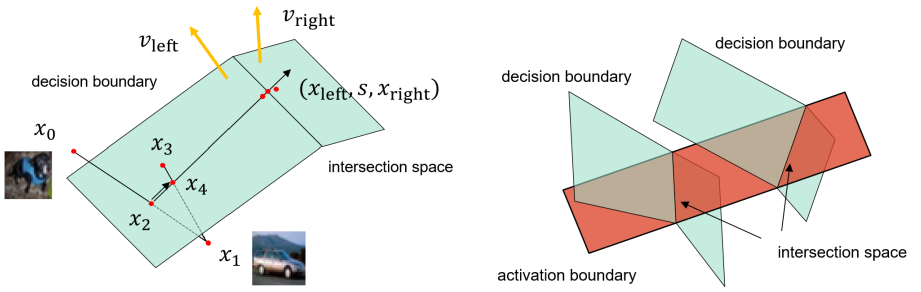
In particular, we see that $\mathbf{F}_{\boldsymbol{x}}^{(L)} = \mathbf{F}_{\boldsymbol{x}}$ and $\boldsymbol{\beta}_{\boldsymbol{x}}^{(L)} = \boldsymbol{\beta}_{\boldsymbol{x}}$, where $\mathbf{D}_{\boldsymbol{x}}^{(L)}$ is regarded as the identity matrix $\mathbf{I}$.

For a fixed activation pattern, we define the maximal subset of the input space on which the pattern remains unchanged as a linear region. When two linear regions have activation patterns that differ in exactly one entry, we call them adjacent and define their separating surface as an activation boundary. As illustrated in Fig. 2, the input space is partitioned into linear regions, and the network acts as an affine transformation on each region. The red and blue lines denote activation boundaries of the first and second layers, respectively. The latter bend where they intersect the former, but remain linear within each linear region. The black line denotes the decision boundary, which bends when it crosses an activation boundary.

Adversarial Capabilities and Objectives. Following [4], we assume the attacker's capabilities and objectives as follows. The attacker can observe the outputs when adaptively chosen queries $\boldsymbol{x}$ are provided to an oracle $\mathcal{O}$, where the oracle is the target hard-label DNN $\hat{y} \circ f$. The attacker's objective is to recover parameters $\hat{\theta}$ that are identical to the original model parameters θ (up to equivalent transformations that do not alter the model's input-output behavior), using sufficiently high-precision floating-point arithmetic. Following prior work, we also make the following assumptions. **Knowledge of the architecture:** The attacker knows the number of layers and units in the model. **Full-domain inputs:** The attacker can provide arbitrary inputs in $\mathbb{R}^{d_0}$. **Precise computations:** The DNN is executed with sufficiently high-precision floating-point arithmetic. **Fully connected ReLU network:** The target model is a multilayer perceptron (MLP) consisting entirely of fully connected layers, and all activation functions are ReLU.

3 Model Extraction in EC25

In this section, we explain the existing model extraction attack proposed by Carlini et al. In this attack, the target model is a hard-label ReLU network. Thus, the only information available to the attacker is the decision boundary at which the classification label changes. The decision boundary of a ReLU network is composed of piecewise linear regions. The attacker traverses along this boundary in a straight manner; however, there are locations at which the boundary bends. We refer to such points as *intersection spaces*. At these points, the activation status of certain neurons in the ReLU network changes, which corresponds to crossing an activation boundary.



(a) Method for finding intersection spaces (Section 3.1).

(b) Consistent intersection spaces are contained in the same activation boundary (Section 3.2).

Fig. 3: Intersection point search and signature recovery.

Definition 3 (Intersection space and intersection point). *For a ReLU network, an intersection of an activation boundary and a decision boundary is referred to as an intersection space. Any point contained in an intersection space is called an intersection point.*

The attack mainly consists of three steps. First, an attacker collects many intersection points and intersection spaces by using the piecewise linearity of the decision boundary. Second, a scalar multiple of the weight vector, denoted by *signature*, is recovered using collected intersection spaces. Finally, the sign of the weight vector and the bias are recovered.

3.1 Collecting Intersection Points

We first collect the intersection points, as outlined in Fig. 3(a).

Step 1: Find a point $\boldsymbol{x}_2$ on the decision boundary. We sample two random inputs $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathbb{R}^{d_0}$ with $\hat{y}(f(\boldsymbol{x}_0)) \neq \hat{y}(f(\boldsymbol{x}_1))$ and use binary search between them to find a point $\boldsymbol{x}_2$ on the decision boundary. Here, $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are drawn from an appropriate distribution (e.g., a Gaussian distribution).

Step 2: Find another point $\boldsymbol{x}_4$ on the decision boundary. We move away from $\boldsymbol{x}_2$ in a random direction to obtain a new point $\boldsymbol{x}_3$. Without loss of generality, assume that $\hat{y}(f(\boldsymbol{x}_3)) = \hat{y}(f(\boldsymbol{x}_0))$; otherwise swap $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. A binary search between $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ then yields another point $\boldsymbol{x}_4$ on the decision boundary.

Step 3: Move along the decision boundary until an intersection point $\boldsymbol{s} \in \mathbb{R}^{d_0}$ is found. Set $\Delta\boldsymbol{x} := \boldsymbol{x}_4 - \boldsymbol{x}_2$ and move along the boundary via $\boldsymbol{x}_\alpha := \boldsymbol{x}_2 + \alpha\Delta\boldsymbol{x}$. For sufficiently small α, $\boldsymbol{x}_\alpha$ remains on the decision boundary. Once $\boldsymbol{x}_\alpha$ leaves the boundary, the path has crossed an activation boundary. We denote the crossing point by $\boldsymbol{s}$ and call it an intersection point.

Step 4: Re-locate the decision boundary. After crossing the activation boundary, we project the point back onto the decision boundary to obtain $\boldsymbol{x}_{\text{right}}$. Let $\boldsymbol{x}_{\text{left}}$ denote the point $\boldsymbol{x}_\alpha$ immediately before the crossing. Then, we have collected a triplet of points: $(\boldsymbol{x}_{\text{left}}, \boldsymbol{s}, \boldsymbol{x}_{\text{right}})$.

Step 5: Find the normal vectors $\boldsymbol{v}_{\text{left}}$ and $\boldsymbol{v}_{\text{right}}$. Let $\boldsymbol{x} \in \mathbb{R}^{d_0}$ be a point on the decision boundary, and let $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\epsilon}$ be a slight perturbation off the boundary. For each standard basis vector $\boldsymbol{e}_i$, find α_i such that $\boldsymbol{x}' + \alpha_i \boldsymbol{e}_i$ lies on the decision boundary. Then normalizing $(1/\alpha_1, \ldots, 1/\alpha_{d_0})^\top$ yields a unit normal vector of the decision boundary at $\boldsymbol{x}$. Applying the same procedure to $\boldsymbol{x}_{\text{left}}$ and $\boldsymbol{x}_{\text{right}}$ yields $\boldsymbol{v}_{\text{left}}$ and $\boldsymbol{v}_{\text{right}}$.

Step 6: Describe intersection space. We can express an intersection space $\mathcal{S}$ of the intersection point $\boldsymbol{s}$ as $\boldsymbol{s} + \text{span}\{\boldsymbol{v}_{\text{left}}, \boldsymbol{v}_{\text{right}}\}^\perp \subset \mathbb{R}^{d_0}$. Set an orthonormal basis of $\text{span}\{\boldsymbol{v}_{\text{left}}, \boldsymbol{v}_{\text{right}}\}^\perp$ as $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_{d_0-2} \subset \mathbb{R}^{d_0}$. Let $\mathbf{N} \in \mathbb{R}^{d_0 \times (d_0-2)}$ denote the matrix obtained by concatenating $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_{d_0-2}$. Then, the intersection space $\mathcal{S}$ can be represented by $(\boldsymbol{s}, \mathbf{N}) \in \mathbb{R}^{d_0} \times \mathbb{R}^{d_0 \times (d_0-2)}$, where $\boldsymbol{s}$ is an arbitrary point in $\mathcal{S}$ and $\mathbf{N}$ is the matrix whose columns form an orthonormal basis for the linear subspace that specifies its direction.

Step 7: Repeat these steps. By sufficiently repeating the above six steps, we collect a large number of intersection spaces $\mathcal{S}_1, \ldots, \mathcal{S}_N$.

3.2 Recovering the Signature

The weight of a neuron corresponds to the normal vector of its associated activation boundary. When multiple intersection spaces are contained in the same activation boundary, as shown in Fig. 3(b), the activation boundary can be reconstructed. By computing its normal vector, we obtain a scalar multiple of the corresponding neuron's weight, which we refer to as its *signature*. In this subsection, we describe the procedure for recovering the signature.

Definition 4 (Signature). *Let $\boldsymbol{w}_k^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$ be the weight of the kth neuron in the ℓth layer, which is equal to the transpose of kth row vector of the weight matrix $\mathbf{W}^{(\ell)}$. The signature of the neuron is a vector $\alpha \cdot \boldsymbol{w}_k^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$ for some $\alpha \in \mathbb{R} \backslash \{0\}$.*

The recovery of the signature is carried out in two main stages. The first stage is the consistency check, `IsConsistent`() (Algorithm 1), and the second is `RecoverSignature`() (Algorithm 2). In the consistency check, we verify whether two intersection spaces belong to the same activation boundary. Once a sufficient number of intersection spaces belonging to the same activation boundary have been collected, the activation boundary is reconstructed from this information, and the signature recovery is performed. Note that we assume that $\mathbf{F}_{\boldsymbol{x}}^{(0)} = \mathbf{I}_{d_0}$ for all $\boldsymbol{x} \in \mathbb{R}^{d_0}$, where $\mathbf{I}_{d_0}$ is the identity matrix of size d_0.

Algorithm 1 `IsConsistent`$(\mathcal{S}_1, \mathcal{S}_2, \{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(\ell-1)}\})$

Input: $\mathcal{S}_1, \mathcal{S}_2$: intersection spaces, $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(\ell-1)}$: recovered weights
Output: False if the $\mathcal{S}_1, \mathcal{S}_2$ are not on the same activation boundary, otherwise True.

Let $\mathbf{N}_1, \mathbf{N}_2$ be basis matrices and $\boldsymbol{s}_1, \boldsymbol{s}_2$ intersection points, corresponding to $\mathcal{S}_1, \mathcal{S}_2$, respectively
Let $\mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}, \mathbf{F}_{\boldsymbol{s}_2}^{(\ell-1)}$ denote the forward local linear matrices.
Let ν be the number of neurons active in either $\boldsymbol{s}_1$ or $\boldsymbol{s}_2$. ▷ If $\ell = 1$, then $\nu = d_0$.
if $\dim(\mathrm{Im}(\mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}\mathbf{N}_1) + \mathrm{Im}(\mathbf{F}_{\boldsymbol{s}_2}^{(\ell-1)}\mathbf{N}_2)) < \nu$ **then**
    return True
else
    return False

Algorithm 2 `RecoverSignature`$(\{\mathcal{S}_1, \dots, \mathcal{S}_m\}, \{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(\ell-1)}\})$

Input: $\mathcal{S}_1, \dots, \mathcal{S}_m$: consistent intersection spaces corresponding to the kth neuron in the ℓth layer, $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(\ell-1)}$: recovered weights
Output: $\alpha \cdot \boldsymbol{w}_k^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$: signature of the neuron

Let $\mathbf{N}_1, \dots, \mathbf{N}_m$ be basis matrices and $\boldsymbol{s}_1, \dots, \boldsymbol{s}_m$ intersection points, corresponding to $\mathcal{S}_1, \dots, \mathcal{S}_m$, respectively
Set $\mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}, \dots, \mathbf{F}_{\boldsymbol{s}_m}^{(\ell-1)}$ be forward local linear matrices.
if $\dim(\mathrm{Im}(\mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}\mathbf{N}_1) + \cdots + \mathrm{Im}(\mathbf{F}_{\boldsymbol{s}_m}^{(\ell-1)}\mathbf{N}_m)) < d_{\ell-1} - 1$ **then**
    return $\perp$ ▷ Insufficient number of intersection spaces.
else
    Find $w \in \mathbb{R}^{d_{\ell-1}}$ such that $\mathrm{span}\{w, \mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}\mathbf{N}_1, \dots, \mathbf{F}_{\boldsymbol{s}_m}^{(\ell-1)}\mathbf{N}_m\} = \mathbb{R}^{d_{\ell-1}}$
    return w

Case 1: The first layer. We begin by considering the case of the first layer. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be intersection spaces, and $\mathbf{N}_1, \mathbf{N}_2$ corresponding bases. If the two intersection spaces are contained in the same activation boundary, it holds that $\dim \mathrm{span}\{\mathbf{N}_1, \mathbf{N}_2\} < d_0$. Here, $\mathrm{span}\{\mathbf{N}_1, \mathbf{N}_2\}$ denotes the subspace of $\mathbb{R}^{d_0}$ spanned by the combined bases contained in $\mathbf{N}_1$ and $\mathbf{N}_2$. The converse of the above statement also holds.

Suppose that a set of consistent intersection spaces has been collected. For two intersection spaces $\mathcal{S}_1$ and $\mathcal{S}_2$ that are confirmed to be consistent, we then perform `RecoverSignature`(). Here, the unique missing dimension corresponding to the defining equation of the target activation boundary represents the signature of the target neuron.

By repeating this procedure sufficiently many times, we obtain the complete collection of signatures of the first layer.

Case 2: Deeper layers. Suppose that the weights $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(\ell-1)}$ up to the $(\ell-1)$st layer have been recovered.

We first perform the consistency check. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be intersection spaces. Denote the corresponding intersection points and basis matrices by $\boldsymbol{s}_1, \boldsymbol{s}_2$ and $\mathbf{N}_1, \mathbf{N}_2$, respectively. Let $\mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}, \mathbf{F}_{\boldsymbol{s}_2}^{(\ell-1)}$ be the forward local linear matrices. If

$\mathcal{S}_1$ and $\mathcal{S}_2$ correspond to the same neuron, the following relation holds:

$$\dim(\mathrm{Im}(\mathbf{F}_{\boldsymbol{s}_1}^{(\ell-1)}\mathbf{N}_1) + \mathrm{Im}(\mathbf{F}_{\boldsymbol{s}_2}^{(\ell-1)}\mathbf{N}_2)) < \nu.$$

where ν is the number of neurons active in either $\boldsymbol{s}_1$ or $\boldsymbol{s}_2$. The converse also holds. We next consider the recovery of the signature. Suppose that, for each neuron, a sufficiently large number of consistent intersection spaces $\mathcal{S}_1, \ldots, \mathcal{S}_m$ have been collected. Then, the signature is recovered by `RecoverSignature`() (Algorithm 2).

3.3 Recovering the Sign

Signature recovery results in a scalar multiple of the weight vector. Since there is an equivalent neural network that uses normalized weight vectors, the scalar scale is not important. However, the sign (i.e., the direction) of the weight vector is still important. The sign affects the input to the ReLU function and, ultimately whether the neuron is active or inactive.

Let $\boldsymbol{s}$ be an intersection point associated with the kth neuron in the ℓth layer. At this point, the pre-activation of the corresponding neuron is 0. Let $\boldsymbol{w}_k^{(\ell)}$ be its weight vector, and suppose we have already obtained a signature $\alpha\boldsymbol{w}_k^{(\ell)}$ and the forward local linear matrix $\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)}$. We then take a sufficiently small displacement $\boldsymbol{\delta} = \mathbf{F}_{\boldsymbol{s}}^{(\ell-1)\top}(\alpha\boldsymbol{w}_k^{(\ell)})$, which represents the normal vector in the input space to the activation boundary of the kth neuron in the ℓth layer. Then,

$$\begin{aligned}\boldsymbol{w}_k^{(\ell)\top}\left(f^{(\ell-1)}(\boldsymbol{s}+\boldsymbol{\delta}) - f^{(\ell-1)}(\boldsymbol{s})\right) &= \boldsymbol{w}_k^{(\ell)\top}\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)}\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)\top}(\alpha\boldsymbol{w}_k^{(\ell)}) \\ &= \alpha\left\|\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)\top}\boldsymbol{w}_k^{(\ell)}\right\|^2.\end{aligned}$$

When $\alpha > 0$, adding and subtracting $\boldsymbol{\delta}$ make the pre-activation positive and negative, respectively. On the other hand, when $\alpha < 0$, adding and subtracting $\boldsymbol{\delta}$ make the pre-activation negative and positive, respectively. Sign recovery is based on the expectation that moving to the positive side would switch the state of neurons in deeper layers more frequently because the ReLU suppresses negative pre-activation.

In this paper, we mainly focus on the difficulty of finding intersection points. Therefore, we omit further details about the sign recovery, and please refer to the original paper [4].

4 Model Extraction in EC25 Is Exponentially Hard

Carlini et al. proposed an algorithm that estimates all parameters of a MLP with ReLU activations using a number of queries polynomial in the number of neurons. In this section, we show that due to the presence of neurons that are always active (which we refer to as *persistent neurons*), it can be difficult to recover all neurons with only polynomially many queries. We first provide a definition of persistent

neurons in Section 4.1, and then, in Section 4.2, empirically demonstrate using models trained on MNIST and Fashion-MNIST (FMNIST) that when Gaussian-distributed vectors are used as inputs, there exist neurons whose activation states are practically impossible to switch without employing an exponentially large number of queries with respect to the depth of the network. In Section 4.3, we further show that the existence of persistent neurons introduces errors in the recovery of deeper-layer neurons, even if computations are assumed to be carried out with arbitrary precision. In Section 5, we present an algorithm that successfully recovers neurons even in the presence of persistent neurons.

4.1 Dead Neurons and Persistent Neurons

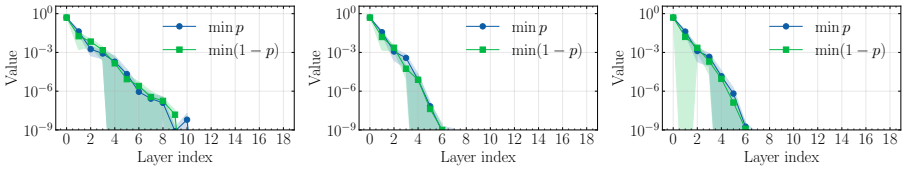
The algorithm proposed by Carlini et al. requires at least two intersection points for each neuron to recover the corresponding weights. In their method, input samples $\{\boldsymbol{x}_i\}_{i=1}^{N}$ are drawn from a suitable input distribution $p_{\boldsymbol{x}}$ (e.g., a Gaussian distribution) and fed into the model. For each pair of inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ that produce different classification outcomes, a binary search is conducted to estimate the decision boundary, which is then used to locate an intersection point. Consequently, if the distribution $p_{\boldsymbol{x}}$ does not yield samples that fall into both the active and inactive regions of each neuron, the attack becomes infeasible. This observation suggests that in Carlini et al.'s algorithm, neurons with extremely low activation or inactivation probabilities may hinder the attack.

To capture such cases, we define neurons with low activation or inactivation probabilities as dead neurons and persistent neurons, respectively:

Definition 5 (Dead neuron and persistent neuron). *Let $p_{\boldsymbol{x}}$ be a probability density function over the input space $\mathbb{R}^{d_0}$, and let $\varepsilon \in [0,1]$ be a positive real number. A neuron whose activation probability is at most ε (i.e., for layer ℓ and neuron index i, $\Pr(\sigma(\boldsymbol{w}_i^{(\ell)\top} f^{(\ell-1)}(\boldsymbol{x}) + b_i^{(\ell)}) > 0) \leq \varepsilon$) is called a $(p_{\boldsymbol{x}}, \varepsilon)$-dead neuron. Similarly, a neuron whose inactivation probability is at most ε (i.e., $\Pr(\sigma(\boldsymbol{w}_i^{(\ell)\top} f^{(\ell-1)}(\boldsymbol{x}) + b_i^{(\ell)}) = 0) \leq \varepsilon$) is called a $(p_{\boldsymbol{x}}, \varepsilon)$-persistent neuron.*

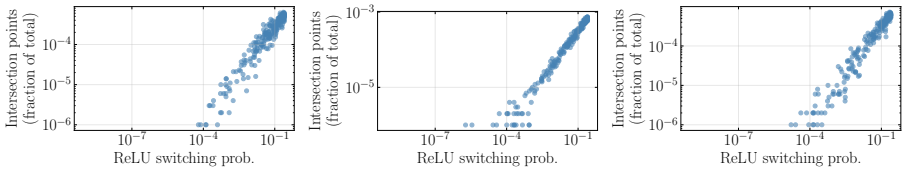
When the input distribution $p_{\boldsymbol{x}}$ is clear from context, we simply refer to these as ε-dead neurons and ε-persistent neurons. Furthermore, when ε itself is also fixed, we omit it and use the terms dead neuron and persistent neuron.

If the adversary's query budget is limited to N, then neurons with $\varepsilon < 1/N$ are unlikely to switch between active and inactive states within the available queries. In such cases, Carlini et al.'s algorithm will, with high probability, fail to recover the weights of these neurons. This stands in contrast to the original assumption of their algorithm, where it is expected that the query budget is sufficient to change the activation state of every neuron. Failure to estimate the weights of dead or persistent neurons can propagate errors to the estimation of neurons in subsequent layers. In particular, if the attack proceeds under the assumption that the weights of such neurons are zero, dead neurons pose no issue since their outputs vanish. However, persistent neurons can have a significant impact on the final output, leading to severe consequences. While the effects will



(a) Untrained model. (b) MNIST. (c) FMNIST.

Fig. 4: Layer-wise minimum neuron activation (or inactivation) probabilities.



(a) Untrained model. (b) MNIST. (c) FMNIST.

Fig. 5: Switching probability versus the proportion of discovered intersection points in the 10th layers of the trained MLPs.

be discussed in Section 4.3, we first empirically investigate the existence of dead and persistent neurons in each layer of neural networks in the next subsection.

4.2 Exponential Queries Required for Persistent Neurons

In this subsection, we empirically demonstrate that, in trained models, there exist neurons whose activation or inactivation probabilities decrease exponentially with respect to the network depth. Moreover, we show that there is a strong proportional relationship between the probability of neuron state switching and the number of intersection points discovered by the algorithm of Carlini et al. These experimental results indicate that, as the depth increases, recovering the weights of dead or persistent neurons becomes exponentially difficult.

Number of dead and persistent neurons when varying ε. We begin by describing the experimental setup. The datasets used were MNIST and Fashion-MNIST (FMNIST), and we trained a 20-layer MLP. Each hidden layer consisted of 256 units, with weights initialized using Kaiming normal initialization [12] and all biases set to zero. The optimizer was Adam [14] with a learning rate of 1×10^{-5}, and the number of training epochs was set to 30.

Let $p_i^{(\ell)}$ denote the activation probability of the ith neuron in layer ℓ. For each layer, we computed both the minimum activation probability (i.e., $\min_i p_i^{(\ell)}$) and the minimum inactivation probability (i.e., $\min_i(1-p_i^{(\ell)})$). These results are shown in Fig. 4. For each dataset, we trained five models with different random seeds, and we report the mean and standard deviation of the minimum probability values across models. The average test accuracies of the trained models are

94.7% for MNIST and 84.8% for FMNIST. For reference, we also show results for an untrained, randomly initialized model in Fig. 4(a), where activation patterns were measured over 10^9 input points sampled from a Gaussian distribution with mean 0 and variance 1. As the figures indicate, there exist neurons whose activation or inactivation probabilities become exponentially small with increasing network depth. When an empirical probability is zero under the finite number of samples, the corresponding point is not displayed on the logarithmic scale; such a missing point means that the probability is below our experimental resolution, not that the true probability is mathematically zero.

Furthermore, we explored 10^6 intersection points and examined the relationship between the discovery rate of intersection points for each neuron, which is defined as the number of intersection points discovered for that neuron divided by the total number of explored intersection points, and the corresponding switching probability $2p_i^{(\ell)}(1-p_i^{(\ell)})$. Fig. 5 plots these quantities for the 10th layer of the trained models, where the x-axis shows the switching probability and the y-axis shows the discovery rate. The experimental results for all the layers are shown in Figs. 10 to 12. The results clearly demonstrate a strong proportional relationship between them. In other words, neurons that are sufficiently close to being dead or persistent (i.e., with extremely small switching probability ε) are unlikely to yield intersection points. Taken together, these results indicate that, in deeper layers, discovering intersection points for neurons with extremely high activation or inactivation probabilities becomes exponentially difficult.

4.3 Analyzing Estimation Errors Induced by Persistent Neuron

In the previous subsection, we empirically observed that recovering persistent neurons becomes exponentially harder as the network depth increases. In other words, as layers grow, an adversary may need exponentially more queries to recover all neurons. Consequently, to run the algorithm of Carlini et al. in polynomial time with respect to the number of neurons, the estimation of dead or persistent neurons with sufficiently small ε must be disregarded.

In the following subsections, we analyze the error that arises when estimating neurons in deeper layers under the condition that at least one persistent neuron has not been recovered in an earlier layer. We show that this leads to an additional unavoidable error in neuron signature estimation. This implies that if even a single persistent neuron is missed in an earlier layer, the resulting estimation error propagates to subsequent layers and may cause the attack to fail. Here, we assume that consistency checks can still be performed.[7]

Effect of Persistent Neurons in Existing Methods. We begin by recalling the neuron estimation algorithm and examining the influence of the presence of

[7] In practice, the presence of persistent neurons may also break consistency. However, in this analysis we assume that consistency checks remain valid and focus only on the additional noise introduced into neuron recovery. If neuron recovery fails, it ultimately indicates that existing attacks are unsuccessful.

a persistent neuron on the estimation. In particular, when a persistent neuron exists, the attacker is forced to treat it as dead, and therefore cannot recover the corresponding coordinate. Consequently, the signature recovery problem in the next layer reduces to an estimation problem with that coordinate removed. We formalize this relationship below.

Suppose an adversary has extracted m intersection points $(\boldsymbol{s}_j)_{j\in[m]}$ corresponding to neuron $\boldsymbol{w}_i^{(\ell)}$. Let $\boldsymbol{v}_j$ and $\boldsymbol{v}'_j$ be the two normal vectors of the jth intersection point. Then, the correct forward matrix up to layer $\ell-1$ is given by $\mathbf{F}_j^{(\ell-1)} = \mathbf{D}_j^{(\ell-1)}\mathbf{W}^{(\ell-1)}\mathbf{D}_j^{(\ell-2)}\mathbf{W}^{(\ell-2)}\cdots\mathbf{D}_j^{(1)}\mathbf{W}^{(1)}$. Here, we just write $\mathbf{F}_j^{(\ell-1)}$ instead of $\mathbf{F}_{\boldsymbol{s}_j}^{(\ell-1)}$ for simplicity. Since $\mathbf{N}_j$ denotes a matrix whose columns form an orthonormal basis of the orthogonal complement $\mathrm{span}\{\boldsymbol{v}_j, \boldsymbol{v}'_j\}^\perp$, we have $(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)^\top \boldsymbol{w}_i^{(\ell)} = \mathbf{0}$. Equivalently, $\boldsymbol{w}_i^{(\ell)\top}(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)^\top \boldsymbol{w}_i^{(\ell)} = 0$. Since this must hold for all $j \in [m]$, it follows that $\boldsymbol{w}_i^{(\ell)\top}\sum_j(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)^\top \boldsymbol{w}_i^{(\ell)} = 0$. Therefore, when m is sufficiently large, the eigenvector corresponding to the minimum eigenvalue (zero) of $\sum_j(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)(\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j)^\top$ coincides with the neuron's signature. This forms the basis of neuron extraction.

Now, suppose that the kth neuron in layer $\ell-1$ is persistent and cannot be recovered. In this case, we approximate its weight and bias as zero, which is equivalent to setting the corresponding diagonal entry of the activation matrix to zero. Thus, the modified diagonal matrix is given by $\tilde{\mathbf{D}}_j^{(\ell-1)} = \mathbf{D}_j^{(\ell-1)} - \boldsymbol{e}_k\boldsymbol{e}_k^\top$. Accordingly, the incorrect forward matrix becomes

$$\begin{aligned}\tilde{\mathbf{F}}_j^{(\ell-1)} &= \tilde{\mathbf{D}}_j^{(\ell-1)}\mathbf{W}^{(\ell-1)}\mathbf{D}_j^{(\ell-2)}\mathbf{W}^{(\ell-2)}\cdots\mathbf{D}_j^{(1)}\mathbf{W}^{(1)}\\ &= \mathbf{F}_j^{(\ell-1)} - \boldsymbol{e}_k\boldsymbol{e}_k^\top\mathbf{W}^{(\ell-1)}\mathbf{D}_j^{(\ell-2)}\cdots\mathbf{D}_j^{(1)}\mathbf{W}^{(1)}.\end{aligned}$$

To simplify notation, for each index j, we denote the output corresponding to the k-th neuron at layer $\ell-1$ after applying the local linear map to $\mathbf{N}_j$ by $\boldsymbol{z}_{k,j}^{(\ell-1)\top} := \boldsymbol{w}_k^{(\ell-1)\top}\mathbf{F}_j^{(\ell-2)}\mathbf{N}_j$. Since $\mathbf{N}_j$ is a matrix consisting of d_0-2 orthonormal basis vectors, the vector $\boldsymbol{z}_{k,j}^{(\ell-1)}$ has d_0-2 entries. More generally, let $\boldsymbol{z}_{r,j}^{(\ell-1)\top}$ denote the rth row of $\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j$. Using this notation, after multiplying by $\mathbf{N}_j$, we can rewrite $\tilde{\mathbf{F}}_j^{(\ell-1)}\mathbf{N}_j = \mathbf{F}_j^{(\ell-1)}\mathbf{N}_j - \boldsymbol{e}_k\,\boldsymbol{z}_{k,j}^{(\ell-1)\top}$, where $\boldsymbol{e}_k$ is the k-th standard basis vector. In addition, the correct local linear mapping can be expressed as $\mathbf{F}_j^{(\ell-1)}\mathbf{N}_j = \left(\boldsymbol{z}_{1,j}^{(\ell-1)}, \boldsymbol{z}_{2,j}^{(\ell-1)}, \ldots, \boldsymbol{z}_{d_{\ell-1},j}^{(\ell-1)}\right)^\top$. When a persistent neuron exists at index k, the (erroneous) local linear mapping becomes $\tilde{\mathbf{F}}_j^{(\ell-1)}\mathbf{N}_j = \left(\boldsymbol{z}_{1,j}^{(\ell-1)}, \ldots, \boldsymbol{z}_{k-1,j}^{(\ell-1)}, \mathbf{0}, \boldsymbol{z}_{k+1,j}^{(\ell-1)}, \ldots, \boldsymbol{z}_{d_{\ell-1},j}^{(\ell-1)}\right)^\top$.

An attacker uses this erroneous matrix to estimate the signature by solving the following minimization problem for the corresponding quadratic form:

$$\min_{\substack{\|\boldsymbol{w}\|=1\\(\boldsymbol{w})_k=0}} \boldsymbol{w}^\top \sum_j \Big((\tilde{\mathbf{F}}_j^{(\ell-1)}\mathbf{N}_j)\,(\tilde{\mathbf{F}}_j^{(\ell-1)}\mathbf{N}_j)^\top\Big)\boldsymbol{w}, \tag{1}$$

where $(\boldsymbol{w})_k$ is the kth coordinate of vector $\boldsymbol{w}$. The constraint $(\boldsymbol{w})_k = 0$ excludes the trivial missing-coordinate direction. Indeed, if the quadratic form were min-

imized over all unit vectors in $\mathbb{R}^{d_{\ell-1}}$, the vector $\boldsymbol{e}_k$ would attain value zero, because the kth row and column are zero. This vector is not a candidate signature, but only represents the coordinate of the unrecovered persistent neuron.

Since the k-th row of $\tilde{\mathbf{F}}_j^{(\ell-1)}\mathbf{N}_j$ is zero for all j, Eq. (1) depends only on the vector $\boldsymbol{w}_{\setminus k}$ obtained by removing the k-th coordinate of $\boldsymbol{w}$. Therefore, the projected problem is equivalently written as

$$\min_{\|\boldsymbol{w}_{\setminus k}\|=1} \ \boldsymbol{w}_{\setminus k}^\top \sum_j \mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j \left(\mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j\right)^\top \boldsymbol{w}_{\setminus k}, \tag{2}$$

where $\mathbf{F}_{j,\setminus k}^{(\ell-1)}$ is obtained by removing the k-th row of $\mathbf{F}_j^{(\ell-1)}$. If a persistent neuron $\boldsymbol{w}_k^{(\ell-1)}$ exists in the previous layer, then its weight and bias cannot be identified by the attacker and are therefore effectively treated as zero. In this case, estimating the signature of a neuron in the next layer is equivalent to using the local linear mapping with the corresponding row removed, namely $\mathbf{F}_{j,\setminus k}^{(\ell-1)}$. Accordingly, the estimated signature also excludes the corresponding coordinate, and the optimization is carried out over a $(d_{\ell-1}-1)$-dimensional vector $\boldsymbol{w}_{\setminus k}$ instead of the original full-dimensional signature vector. Consequently, the attacker's objective is no longer to recover the complete signature $\boldsymbol{w}_i^{(\ell)}$, but rather the reduced signature $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ with the k-th component removed.

Discrepancy from the Exact Signature Estimation. By the reduction above, when a persistent neuron exists, signature recovery is equivalent to estimating the signature using the matrix obtained by removing the k-th row from the local linear mapping. Since this matrix differs from the true local linear mapping in the presence of a persistent neuron, substituting the true reduced signature $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ into Eq. (2) does not, in general, yield zero. Consequently, minimizing Eq. (2) does not necessarily recover $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$.

To quantify this discrepancy, we next derive a bound on the angle between the true reduced signature and the signature obtained from Eq. (2). The following theorem provides an explicit upper bound on $\tan\theta$, where θ denotes the angle between the true signature and the signature estimated by Eq. (2).

Theorem 1 (Upper Bound on Signature Discrepancy). *Let* $\tilde{\mathbf{G}} = \frac{1}{m}\sum_j \mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j \left(\mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j\right)^\top$ *be the matrix used in the minimization problem Eq.* (2)*, where the normalization by* m *is introduced only for convenience and does not change the solution of Eq.* (2)*. Define* $\mathbf{G} = \tilde{\mathbf{G}} - \boldsymbol{\beta}\boldsymbol{\beta}^\top/\alpha$*, where* $\alpha = \frac{1}{m}\sum_j \boldsymbol{z}_{k,j}^{(\ell-1)\top}\boldsymbol{z}_{k,j}^{(\ell-1)}$*, and*

$$\boldsymbol{\beta} = \frac{1}{m}\sum_j \mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j\,\boldsymbol{z}_{k,j}^{(\ell-1)} = \frac{1}{m}\sum_j (\boldsymbol{z}_{1,j}^{(\ell-1)\top}\boldsymbol{z}_{k,j}^{(\ell-1)}, \boldsymbol{z}_{2,j}^{(\ell-1)\top}\boldsymbol{z}_{k,j}^{(\ell-1)}, \ldots, \boldsymbol{z}_{k-1,j}^{(\ell-1)\top}\boldsymbol{z}_{k,j}^{(\ell-1)}, \boldsymbol{z}_{k+1,j}^{(\ell-1)\top}\boldsymbol{z}_{k,j}^{(\ell-1)}, \ldots, \boldsymbol{z}_{d_{\ell-1},j}^{(\ell-1)\top}\boldsymbol{z}_{k,j}^{(\ell-1)})^\top.$$

Assume that $\tilde{\mathbf{G}} \in \mathbb{R}^{(d_{\ell-1}-1)\times(d_{\ell-1}-1)}$ *has full rank* $d_{\ell-1}-1$*, and that* $\boldsymbol{\beta}$ *is not orthogonal to any eigenvector of* $\tilde{\mathbf{G}}$*. Consider the eigendecomposition* $\mathbf{G} = \sum_\eta \mu_\eta \boldsymbol{q}_\eta \boldsymbol{q}_\eta^\top$*, where the eigenvalues satisfy* $0 = \mu_1 < \mu_2 \le \cdots \le \mu_{d_{\ell-1}-1}$*. Then the eigenvector* $\boldsymbol{q}_1$ *coincides with the true reduced signature* $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$*. In addition, let* θ *be the angle between the signature estimated via Eq.* (2) *and the true reduced signature. Then*

$$\tan\theta \le \frac{|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta}|}{\alpha + \boldsymbol{\beta}^\top \mathbf{G}^\dagger \boldsymbol{\beta}} \sqrt{\sum_{\eta=2} \frac{(\boldsymbol{q}_\eta^\top \boldsymbol{\beta})^2}{(\mu_\eta - \lambda_1)^2}}, \tag{3}$$

where λ_1 *is the smallest eigenvalue of* $\tilde{\mathbf{G}}$ *and* $\mathbf{G}^\dagger = \sum_{\eta=2} \boldsymbol{q}_\eta \boldsymbol{q}_\eta^\top / \mu_\eta$ *denotes the pseudoinverse of* $\mathbf{G}$*.*

Before proving the theorem, we comment on the assumptions used above. First, we assume that $\tilde{\mathbf{G}} \in \mathbb{R}^{(d_{\ell-1}-1)\times(d_{\ell-1}-1)}$ has full rank $d_{\ell-1}-1$. This is the same generic rank condition used in layer-wise extraction: writing $\mathbf{A}_j = \mathbf{F}_{j,\setminus k}^{(\ell-1)} \mathbf{N}_j$, it is equivalent to requiring the stacked matrix $[\mathbf{A}_1, \ldots, \mathbf{A}_m]$ to have full row rank. Although each individual $\mathbf{A}_j$ may be low rank, different intersection points induce different local subspaces, and their concatenation can span the full reduced layer as m increases. We also confirmed this behavior in a simple expanding-network experiment on the Iris dataset, using an MLP with input dimension 4, hidden width 10, and output dimension 3, where $\tilde{\mathbf{G}}$ reached full rank after collecting sufficiently many intersection points. In our analysis, we assume that sufficiently many intersection points are collected so that this condition holds.

We also assume that the vector $\boldsymbol{\beta}$ is not orthogonal to any eigenvector of $\tilde{\mathbf{G}}$. Since neural networks are trained in a stochastic manner and these quantities depend on many continuous parameters, exact orthogonality is highly unlikely in practice, and its probability is expected to be negligible.

Proof. First, we prove that the eigenvector corresponding to the smallest eigenvalue of $\mathbf{G}$ coincides with the true reduced signature $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$. We then derive the stated upper bound on $\tan\theta$.

(1) Proof that $\boldsymbol{q}_1 = \boldsymbol{w}_{i,\setminus k}^{(\ell)}$*.* Consider the matrix $\frac{1}{m}\sum_j \mathbf{F}_j^{(\ell-1)} \mathbf{N}_j \left(\mathbf{F}_j^{(\ell-1)} \mathbf{N}_j\right)^\top$, and permute its rows and columns so that the k-th row and column are moved to the first row and column. Denote the resulting matrix by $\tilde{\mathbf{G}}_{\mathrm{mov}} = \begin{pmatrix} \alpha & \boldsymbol{\beta}^\top \\ \boldsymbol{\beta} & \tilde{\mathbf{G}} \end{pmatrix}$. For this matrix, a zero-eigenvalue eigenvector is given by $(w_{i,k}^{(\ell)}, \boldsymbol{w}_{i,\setminus k}^{(\ell)\top})^\top$, where $w_{i,k}^{(\ell)}$ denotes the k-th component of the signature of the i-th neuron at layer ℓ. Hence, $\tilde{\mathbf{G}}_{\mathrm{mov}}(w_{i,k}^{(\ell)}, \boldsymbol{w}_{i,\setminus k}^{(\ell)\top})^\top = 0$, which implies

$$\begin{cases} \alpha\, w_{i,k}^{(\ell)} + \boldsymbol{\beta}^\top \boldsymbol{w}_{i,\setminus k}^{(\ell)} = 0, \\ \boldsymbol{\beta}\, w_{i,k}^{(\ell)} + \tilde{\mathbf{G}}\, \boldsymbol{w}_{i,\setminus k}^{(\ell)} = 0. \end{cases}$$

Solving the first equation gives $w_{i,k}^{(\ell)} = -\boldsymbol{\beta}^\top \boldsymbol{w}_{i,\setminus k}^{(\ell)}/\alpha$. Substituting this into the second equation yields $\left(\tilde{\mathbf{G}} - \boldsymbol{\beta}\boldsymbol{\beta}^\top/\alpha\right)\boldsymbol{w}_{i,\setminus k}^{(\ell)} = 0$. Thus, by definition of $\mathbf{G}$, the vector $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ is the eigenvector of $\mathbf{G}$ associated with eigenvalue zero.

(2) Representation of eigenvalues of $\tilde{\mathbf{G}}$. To derive the bound on $\tan\theta$, consider an eigenpair $(\lambda, \boldsymbol{p})$ of $\tilde{\mathbf{G}}$ with λ not an eigenvalue of $\mathbf{G}$. Then from $\tilde{\mathbf{G}}\boldsymbol{p} = \lambda\boldsymbol{p}$ and $\tilde{\mathbf{G}} = \mathbf{G} + \boldsymbol{\beta}\boldsymbol{\beta}^\top/\alpha$, we obtain $(\mathbf{G} - \lambda\mathbf{I})\boldsymbol{p} = -(\boldsymbol{\beta}^\top\boldsymbol{p})\boldsymbol{\beta}/\alpha$. Since $\mathbf{G} - \lambda\mathbf{I}$ is invertible, this gives $\boldsymbol{p} = -(\boldsymbol{\beta}^\top\boldsymbol{p})(\mathbf{G} - \lambda\mathbf{I})^{-1}\boldsymbol{\beta}/\alpha$, and hence $\boldsymbol{p} \propto (\mathbf{G} - \lambda\mathbf{I})^{-1}\boldsymbol{\beta}$, where $\propto$ denotes equality up to a nonzero scalar factor. Left-multiplying by $\boldsymbol{\beta}^\top$ and dividing by $\boldsymbol{\beta}^\top\boldsymbol{p}$ yields

$$1 + \boldsymbol{\beta}^\top(\mathbf{G} - \lambda\mathbf{I})^{-1}\boldsymbol{\beta}/\alpha = 0. \tag{4}$$

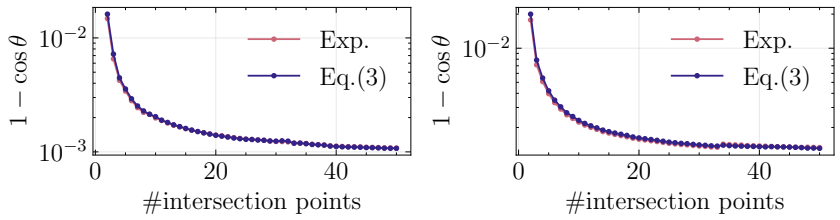
(3) Upper bound on λ_1. Using the eigendecomposition of $\mathbf{G}$, we obtain $\boldsymbol{\beta}^\top(\mathbf{G} - \lambda\mathbf{I})^{-1}\boldsymbol{\beta} = -\frac{(\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta})^2}{\lambda_1} + \sum_{\eta=2}\frac{(\boldsymbol{q}_\eta^\top\boldsymbol{\beta})^2}{\mu_\eta - \lambda_1}$. Combining with Eq. (4) gives $\alpha - \frac{(\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta})^2}{\lambda_1} + \sum_{\eta=2}\frac{(\boldsymbol{q}_\eta^\top\boldsymbol{\beta})^2}{\mu_\eta - \lambda_1} = 0$. Let $\mathbf{G}^\dagger = \sum_{\eta=2}\boldsymbol{q}_\eta\boldsymbol{q}_\eta^\top/\mu_\eta$. Since the last sum is bounded below by replacing $\mu_\eta - \lambda_1$ with μ_η, we obtain

$$\lambda_1 \le \frac{(\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta})^2}{\alpha + \boldsymbol{\beta}^\top\mathbf{G}^\dagger\boldsymbol{\beta}}. \tag{5}$$

(4) Upper bound on $\tan\theta$. From step (2), the eigenvector associated with λ_1 can be written as $\hat{\boldsymbol{p}}_1 = -\frac{\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta}}{\lambda_1}\boldsymbol{w}_{i,\setminus k}^{(\ell)} + \sum_{\eta=2}\frac{\boldsymbol{q}_\eta^\top\boldsymbol{\beta}}{\mu_\eta - \lambda_1}\boldsymbol{q}_\eta$. From this decomposition, $\tan\theta = \left\|\sum_{\eta=2}\frac{\boldsymbol{q}_\eta^\top\boldsymbol{\beta}}{\mu_\eta - \lambda_1}\boldsymbol{q}_\eta\right\| \Big/ \left|\frac{\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta}}{\lambda_1}\right|$. Substituting Eq. (5) yields Eq. (3). □

Discussion on Theorem 1. Based on Theorem 1, we explain why the estimation error is difficult to avoid within the existing method. The right-hand side of Eq. (3) is the product of the inner-product factor $|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta}|$ and the factor $(\alpha + \boldsymbol{\beta}^\top\mathbf{G}^\dagger\boldsymbol{\beta})^{-1}\sqrt{\sum_{\eta=2}(\boldsymbol{q}_\eta^\top\boldsymbol{\beta})^2/(\mu_\eta - \lambda_1)^2}$. Thus, if $|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta}| = 0$, then the bound gives $\tan\theta = 0$. The second factor can also vanish, but this requires $\boldsymbol{\beta}$ to have no component in the directions $\boldsymbol{q}_\eta$ for $\eta \ge 2$, i.e., $\boldsymbol{\beta}$ must be aligned with the true reduced signature $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$. These are special alignments, and we do not expect them to occur in general. Since Eq. (3) provides only an upper bound on the error, these conditions are sufficient but not necessarily necessary. However, as shown empirically in Section 4.4, the bound closely matches the observed values of $\tan\theta$, which indicates that these factors explain the observed estimation error.

The role of the inner-product factor can also be understood from the fact, shown in Theorem 1, that the matrix that yields the true reduced signature is not $\tilde{\mathbf{G}} = \frac{1}{m}\sum_j \mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j(\mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j)^\top$, but rather $\mathbf{G} = \tilde{\mathbf{G}} - \boldsymbol{\beta}\boldsymbol{\beta}^\top/\alpha$. If the true



(a) MNIST. (b) FMNIST.

Fig. 6: Geometric average of $1-\cos\theta$ between the true and estimated signatures for models with a persistent neuron at width 256.

reduced signature $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ is orthogonal to $\boldsymbol{\beta}$, then $\tilde{\mathbf{G}}\,\boldsymbol{w}_{i,\setminus k}^{(\ell)} = (\mathbf{G}+\boldsymbol{\beta}\boldsymbol{\beta}^\top/\alpha)\boldsymbol{w}_{i,\setminus k}^{(\ell)} = 0$, and hence $\tilde{\mathbf{G}}$ still has $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ as an eigenvector associated with eigenvalue 0. In this case, the signature can still be recovered correctly.

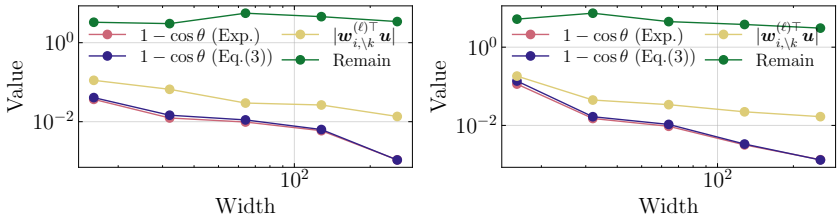
However, it is unlikely that these vectors are exactly orthogonal. Indeed, since $\boldsymbol{\beta} = \frac{1}{m}\sum_j \mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j\,\boldsymbol{z}_{k,j}^{(\ell-1)}$, taking the inner product with the true reduced signature gives $\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{\beta} = \frac{1}{m}\sum_j \boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j\,\boldsymbol{z}_{k,j}^{(\ell-1)}$. Here, $\tilde{\boldsymbol{z}}_{i,j}^{(\ell)} = \boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\mathbf{F}_{j,\setminus k}^{(\ell-1)}\mathbf{N}_j$ can be interpreted as the local variation of the pre-activation of the i-th neuron at layer ℓ computed while ignoring the persistent neuron. Therefore, the above inner product is essentially an average of products between these local variations and the local variations of the k-th neuron in layer $\ell-1$. In a trained network, there is no reason to expect these local variations to be exactly orthogonal on average. Similarly, there is no reason to expect $\boldsymbol{\beta}$ to be exactly aligned with the true reduced signature. Thus, the exceptional cases where the bound vanishes are unlikely in practice.

From these considerations, when a persistent neuron exists, an unavoidable error term is expected to be introduced into the signature estimation. The magnitude of this error is evaluated experimentally in the next subsection.

4.4 Experimental Results of Estimated Errors

We next quantify the noise introduced into neuron estimation by persistent neurons. We show experimentally that the estimated signature contains noise between $O(1/d)$ and $O(1/(d\sqrt{d}))$, where d is the model width, and interpret this scaling using Eq. (3).

Experimental Results. The model training conditions are essentially the same as those in Section 4.2, except that we used five different widths: 16, 32, 64, 128, and 256. For each trained model, we considered the case where one persistent neuron in the 9th layer could not be recovered, and we then attempted to recover neurons in the 10th layer.



(a) MNIST. (b) FMNIST.

Fig. 7: Geometric average of $1-\cos\theta$ between true and estimated signatures with a persistent neuron while varying the model width.

For 10th-layer recovery, we extracted 10^7 intersection points per model and retained only neurons with at least 50 intersection points. We recovered each such weight using 50 intersection points. Since 50 intersection points may not recover all signature dimensions (depending on previous-layer activation patterns), we computed the angle θ to the true signature only on the successfully estimated components.

Fig. 6 shows $1-\cos\theta$ as the number of intersection points varies for a trained width-256 model. "Exp." denotes empirical recovery with a persistent neuron, and "Eq. (3)" denotes the bound on $1-\cos\theta$ obtained from Eq. (3) via $\cos\theta = 1/\sqrt{1+\tan^2\theta}$. The bound closely matches the experiments. Although the error decreases with more intersection points, the decrease gradually saturates, making it difficult to eliminate the error by increasing intersection points alone.

Fig. 7 shows the results for $1-\cos\theta$ as a function of the model width. For each width, we used the number of intersection points that achieved the smallest empirical value of $1-\cos\theta$. To clarify which terms dominate the bound, we plot the following quantities. By normalizing the numerator and denominator of Eq. (3) using $\|\boldsymbol{\beta}\|$, the bound can be rewritten as

$$\tan\theta \le \frac{|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{u}|}{\alpha/\|\boldsymbol{\beta}\|^2 + \boldsymbol{u}^\top \mathbf{G}^\dagger \boldsymbol{u}} \sqrt{\sum_{\eta=2} \frac{(\boldsymbol{q}_\eta^\top \boldsymbol{u})^2}{(\mu_\eta - \lambda_1)^2}}, \tag{6}$$

where $\boldsymbol{u} = \boldsymbol{\beta}/\|\boldsymbol{\beta}\|$. We separately plot $|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{u}|$. The curve labeled "Remain" is $(\alpha/\|\boldsymbol{\beta}\|^2 + \boldsymbol{u}^\top \mathbf{G}^\dagger \boldsymbol{u})^{-1}\sqrt{\sum_{\eta=2} \frac{(\boldsymbol{q}_\eta^\top \boldsymbol{u})^2}{(\mu_\eta-\lambda_1)^2}}$. For small θ, $1-\cos\theta \approx \tan^2\theta/2$, so $1-\cos\theta$ is approximately determined by the product of these two factors; on log-log axes, its slope is twice the sum of their slopes. The measured $1-\cos\theta$ closely matches the value predicted by Eq. (6).

From the figure, we observe that the empirical values of $1-\cos\theta$ decrease approximately linearly in the log–log plot. This indicates a power-law dependence on the model width d, and we therefore perform a linear fit in the log–log scale. The resulting slopes are -1.12 for MNIST and -1.51 for FMNIST. These values suggest that, for width d, the estimation error decreases at a rate between $O(1/d)$

and $O(1/(d\sqrt{d}))$. The source of this improvement can also be inferred from the figure: since the "Remain" term is nearly flat across widths, the dominant contribution comes from the decay of $|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{u}|$.

From these experimental results, we conclude that the estimation accuracy saturates even if the number of intersection points is further increased, and that the saturation level scales approximately as $O(1/d)$ to $O(1/(d\sqrt{d}))$ with respect to the model width d.

Discussion. Here, we discuss why the estimation accuracy improves at approximately a rate between $O(1/d)$ and $O(1/(d\sqrt{d}))$ with respect to the width d, based on Eq. (6). Since a precise analysis for fully trained models is difficult, we instead consider a highly simplified setting. Specifically, we assume that the vector $\boldsymbol{\beta}$ (and hence $\boldsymbol{u}$) is independent of $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$, and that $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ is uniformly distributed on the unit sphere. Although this assumption does not necessarily hold for trained models, at least under Kaiming normal initialization each weight is sampled from a Gaussian distribution. Therefore, it is reasonable to assume that the normalized weight vector $\boldsymbol{w}_{i,\setminus k}^{(\ell)}$ is uniformly distributed on the unit hypersphere at initialization.

Under this assumption, when d is sufficiently large, we have approximately $|\boldsymbol{w}_{i,\setminus k}^{(\ell)\top}\boldsymbol{u}| = O(1/\sqrt{d})$. We have experimentally confirmed that the remaining factor $(\alpha/\|\boldsymbol{\beta}\|^2 + \boldsymbol{u}^\top \mathbf{G}^\dagger \boldsymbol{u})^{-1}\sqrt{\sum_{\eta=2} \frac{(\boldsymbol{q}_\eta^\top \boldsymbol{u})^2}{(\mu_\eta - \lambda_1)^2}}$ is nearly independent of the width. This implies that $\tan\theta$ decreases approximately as $O(1/\sqrt{d})$ as the width increases. When θ is sufficiently small, the approximation $1 - \cos\theta \approx \frac{1}{2}\tan^2\theta$ holds, which suggests that $1 - \cos\theta = O(1/d)$. This provides a rough explanation for why the estimation accuracy improves at a rate between $O(1/d)$ and $O(1/(d\sqrt{d}))$. Of course, in practice an attacker cannot control or increase the width of the victim model's neural network architecture. Hence, reducing the error through width scaling is not a viable option for the attacker. From this perspective as well, removing the estimation error is inherently difficult.

5 Cross-Layer Extraction for Persistent Neurons

As discussed above, the presence of persistent neurons prevents their weight vectors from being recovered by existing techniques. An attacker cannot observe the corresponding activation boundaries and therefore cannot extract the associated weights. Unlike dead neurons, persistent neurons affect subsequent layers, which makes recovering their contributions necessary for model extraction. Since intersection points for persistent neurons are not available, we need an alternative method. We recover their contribution via a cross-layer boundary, exploiting decision and activation boundaries of neurons in a deeper layer.

Let $\mathcal{P}$ be the index set of persistent neurons in the ℓth layer. Existing model extraction fails to recover the weight $\boldsymbol{w}_j^{(\ell)}$ for $j \in \mathcal{P}$. On the other hand, when

a point lies on the activation boundary at depth $\ell+1$ and neuron index k, we still have the following relation:

$$-b_k^{(\ell+1)} = \boldsymbol{w}_k^{(\ell+1)\top}\boldsymbol{z}^{(\ell)} = \boldsymbol{w}_{k,\bar{\mathcal{P}}}^{(\ell+1)\top}\boldsymbol{z}_{\bar{\mathcal{P}}}^{(\ell)} + \sum_{j\in\mathcal{P}} w_{k,j}^{(\ell+1)}(\boldsymbol{w}_j^{(\ell)\top}\boldsymbol{z}^{(\ell-1)} + b_j^{(\ell)}),$$
$$= \boldsymbol{w}_{k,\bar{\mathcal{P}}}^{(\ell+1)\top}\boldsymbol{z}_{\bar{\mathcal{P}}}^{(\ell)} + \left(\sum_{j\in\mathcal{P}} w_{k,j}^{(\ell+1)}\boldsymbol{w}_j^{(\ell)}\right)^{\top}\boldsymbol{z}^{(\ell-1)} + \sum_{j\in\mathcal{P}} w_{k,j}^{(\ell+1)} b_j^{(\ell)},$$

We denote by $\boldsymbol{a}_{\mathcal{P}}$ the subvector of $\boldsymbol{a}$ consisting of the components indexed by $\mathcal{P}$, and by $\boldsymbol{a}_{\bar{\mathcal{P}}}$ the subvector consisting of the remaining components. Define

$$\boldsymbol{w}_{cross} = \begin{pmatrix} \boldsymbol{w}_{k,\bar{\mathcal{P}}}^{(\ell+1)} \\ \sum_{j\in\mathcal{P}} w_{k,j}^{(\ell+1)}\boldsymbol{w}_j^{(\ell)} \end{pmatrix}.$$

Then, points on the activation boundary satisfy an affine constraint of the form $\boldsymbol{w}_{cross}^{\top}\begin{pmatrix}\boldsymbol{z}_{\bar{\mathcal{P}}}^{(\ell)} \\ \boldsymbol{z}^{(\ell-1)}\end{pmatrix} = \text{const}$. Equivalently, for any two boundary points, their difference is orthogonal to $\boldsymbol{w}_{cross}$. We collect multiple points on the boundary, take differences against a fixed reference point, and recover $\boldsymbol{w}_{cross}$ up to scale as a normal vector of the subspace spanned by these differences. This vector contains a linear combination of the persistent neuron weights $\boldsymbol{w}_j^{(\ell)}$ for $j \in \mathcal{P}$.

Some important considerations must be made regarding cross-layer extraction. First, it is necessary to modify the corresponding consistency algorithm. While the existing algorithm analyzes each intersection space pairwise, cross-layer extraction requires analyzing multiple intersection spaces simultaneously. Second, cross-layer analysis can extract only the span of the original weight vectors of persistent neurons. More precisely, since persistent neurons are always active and are never suppressed by ReLU, their corresponding signs and biases are linearly combined into the next layer. Unless they become inactive, we cannot extract them individually. Thus, in the recovered model, we need exceptional treatment for them. Specifically, any basis of the span can be used, and the corresponding ReLU must be removed.

5.1 Modified Consistency Algorithm

Let $\boldsymbol{s} \in \mathbb{R}^{d_0}$ be an intersection point, and let $\mathcal{S}$ denote the corresponding intersection space. The goal of the consistency algorithm is to determine whether two intersection spaces $\mathcal{S}_1$ and $\mathcal{S}_2$ are associated with the same activation boundary.

We first recall the existing consistency algorithm. This algorithm takes two intersection spaces $\mathcal{S}_1$ and $\mathcal{S}_2$. If they are associated with the same activation boundary at depth ℓ and index k, the sum of their subspaces, $f_1^{(\ell-1)}(\mathcal{S}_1) + f_2^{(\ell-1)}(\mathcal{S}_2)$, is orthogonal to the weight vector of the corresponding neuron, $\boldsymbol{w}_k^{(\ell)}$. Here, $f_i^{(\ell-1)}$ denotes the local linear map up to depth $\ell-1$ induced by the input $\boldsymbol{s}_i$ at the intersection point. On the other hand, if they are not associated with

the same activation boundary, no such orthogonal vector exists in the sum of subspaces. Thus, we compute the rank of $f_1^{(\ell-1)}(\mathcal{S}_1) + f_2^{(\ell-1)}(\mathcal{S}_2)$, and if a rank deficiency is observed, we identify that the two spaces are associated with the same activation boundary. Specifically, we count the number of neurons active in either $f_1^{(\ell-1)}(\boldsymbol{s}_1)$ or $f_2^{(\ell-1)}(\boldsymbol{s}_2)$. If the observed rank is lower than this number, the two intersection spaces are considered to belong to the same activation boundary.

In cross-layer extraction, the local linear map must be constructed across multiple layers. Instead of using $f^{(\ell-1)}(\boldsymbol{s})$, we introduce

$$g^{(\ell)}(\boldsymbol{s}) = \begin{pmatrix} f_{\bar{\mathcal{P}}}^{(\ell)}(\boldsymbol{s}) \\ f^{(\ell-1)}(\boldsymbol{s}) \end{pmatrix},$$

where $f_{\bar{\mathcal{P}}}^{(\ell)}(\boldsymbol{s})$ represents the output at layer ℓ with the coordinates in $\mathcal{P}$ removed. These outputs are prepended to the original $f^{(\ell-1)}(\boldsymbol{s})$. If $\mathcal{S}_1$ and $\mathcal{S}_2$ are associated with the same activation boundary at depth $\ell+1$ and index k, the sum of subspaces, $g_1^{(\ell)}(\mathcal{S}_1) + g_2^{(\ell)}(\mathcal{S}_2)$, lies on the same activation boundary and is orthogonal to a weight-related vector $\boldsymbol{w}_{cross}$.

$$\boldsymbol{w}_{cross} \;\perp\; \left(g_1^{(\ell)}(\mathcal{S}_1) + g_2^{(\ell)}(\mathcal{S}_2)\right). \tag{7}$$

Unlike the existing consistency algorithm, the cross-layer consistency algorithm requires more than two samples. To understand this behavior, we consider the corresponding local linear forward matrix, $\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)}$. For sufficiently small $\boldsymbol{\delta}$, there is a matrix $\mathbf{M}$ satisfying

$$g^{(\ell)}(\boldsymbol{s}+\boldsymbol{\delta}) = g^{(\ell)}(\boldsymbol{s}) + \mathbf{M}\boldsymbol{\delta}, \qquad \mathbf{M} = \begin{pmatrix} \mathbf{D}_{\bar{\mathcal{P}}}\mathbf{D}^{(\ell)}\mathbf{W}^{(\ell)}\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)} \\ \mathbf{F}_{\boldsymbol{s}}^{(\ell-1)} \end{pmatrix},$$

where $\mathbf{D}_{\bar{\mathcal{P}}}$ denotes a diagonal matrix whose ith diagonal entry is 1 if and only if $i \in \bar{\mathcal{P}}$.

Let $\mathbf{N}_i$ be a basis of the intersection space $\mathcal{S}_i$. Then, the basis of the sum of subspaces is represented by the column-wise concatenation $(\mathbf{M}_1\mathbf{N}_1, \mathbf{M}_2\mathbf{N}_2)$. The orthogonality condition in Eq. (7) can be rewritten as

$$\boldsymbol{w}_{cross}^{\top}\,(\mathbf{M}_1\mathbf{N}_1, \mathbf{M}_2\mathbf{N}_2) = 0,$$

or equivalently,

$$\begin{pmatrix} \mathbf{N}_1^{\top}\mathbf{M}_1^{\top} \\ \mathbf{N}_2^{\top}\mathbf{M}_2^{\top} \end{pmatrix} \boldsymbol{w}_{cross} = 0.$$

In cross-layer extraction, the ranks of $\mathbf{M}_1$ and $\mathbf{M}_2$ are typically low because the lower half of the row vectors are linearly dependent on those in the upper half. Therefore, the dimension of the kernel of $\mathbf{M}_i^{\top}$ is large, and

$$\dim\left(\mathrm{Ker}(\mathbf{M}_1^{\top}) \cap \mathrm{Ker}(\mathbf{M}_2^{\top})\right) > 0$$

Algorithm 3 `IsUnifiedConsistent`($\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m, \mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_m$)

Input: $\mathbf{M}_i$: a matrix to transform to the target layer, $\mathbf{N}_i$: the corresponding intersection space basis.
Output: (*gap*, *score*, $\boldsymbol{v}_1$): statistics and the top generalized eigenvector.

for $i = 1$ to m **do**
  $\mathbf{L}_i \leftarrow \mathbf{M}_i\mathbf{N}_i$
$\mathbf{M} = \sum_i \mathbf{M}_i\mathbf{M}_i^\top$
$\mathbf{L} = \sum_i \mathbf{L}_i\mathbf{L}_i^\top$
Solve a generalized eigenvalue problem, $\mathbf{M}\boldsymbol{v} = \lambda\mathbf{L}\boldsymbol{v}$
gap $= \lambda_1/\lambda_2$ ▷ $\lambda_{\max} = \lambda_1 > \lambda_2 > \cdots$.
score $= (\boldsymbol{v}_1^\top\mathbf{L}\boldsymbol{v}_1)/(\boldsymbol{v}_1^\top\mathbf{M}\boldsymbol{v}_1)$ ▷ $\boldsymbol{v}_1$ is the maximum eigenvector.
return *gap*, *score*, $\boldsymbol{v}_1$

holds. This implies that the intersection of the kernels, $\left(\mathrm{Ker}(\mathbf{M}_1^\top) \cap \mathrm{Ker}(\mathbf{M}_2^\top)\right)$, contains basis vectors orthogonal to $\left(g_1^{(\ell)}(\mathcal{S}_1) + g_2^{(\ell)}(\mathcal{S}_2)\right)$ regardless of the specific choice of $\mathcal{S}_1$ and $\mathcal{S}_2$. In other words, the conventional consistency algorithm always observes rank deficiency[8].

To handle low-rank matrices, we propose a unified consistency algorithm. Algorithm 3 shows the pseudocode. Given matrices $\mathbf{M}_i$ and basis matrices $\mathbf{N}_i$, we first compute the transformed bases $\mathbf{L}_i = \mathbf{M}_i\mathbf{N}_i$. Our goal is to determine whether there exists $\boldsymbol{w}_{cross}$ satisfying $\boldsymbol{w}_{cross}^\top\mathbf{L}_i = \mathbf{0}$ and $\boldsymbol{w}_{cross}^\top\mathbf{M}_i \neq \mathbf{0}$ for all i. To this end, we use

$$\sum_i \|\boldsymbol{w}_{cross}^\top\mathbf{L}_i\|^2 = \boldsymbol{w}_{cross}^\top\left(\sum_i \mathbf{L}_i\mathbf{L}_i^\top\right)\boldsymbol{w}_{cross} = 0,$$
$$\sum_i \|\boldsymbol{w}_{cross}^\top\mathbf{M}_i\|^2 = \boldsymbol{w}_{cross}^\top\left(\sum_i \mathbf{M}_i\mathbf{M}_i^\top\right)\boldsymbol{w}_{cross} > 0.$$

Let $\mathbf{L} = \sum_i \mathbf{L}_i\mathbf{L}_i^\top$ and $\mathbf{M} = \sum_i \mathbf{M}_i\mathbf{M}_i^\top$. Then, maximizing $(\boldsymbol{v}^\top\mathbf{M}\boldsymbol{v})/(\boldsymbol{v}^\top\mathbf{L}\boldsymbol{v})$ leads to a generalized eigenvalue problem $\mathbf{M}\boldsymbol{v} = \lambda\mathbf{L}\boldsymbol{v}$, where the maximum eigenvector $\boldsymbol{v}_1$ achieves the maximum[9]. Specifically, we check two values; the first value is *score* that is $(\boldsymbol{v}_1^\top\mathbf{L}\boldsymbol{v}_1)/(\boldsymbol{v}_1^\top\mathbf{M}\boldsymbol{v}_1)$, and the second value is *gap* that is

[8] Note that this issue can happen in the original setting. For example, if there is a layer with very few active neurons before reaching depth ℓ, $\mathrm{rank}(\mathbf{F}_{\boldsymbol{s}}^{(\ell-1)})$ is bounded by the number of active neurons in that layer. However, in practice, the likelihood of such false positives is significantly lower than in cross-layer scenarios, making conventional simple algorithms sufficient.

[9] We also tried the generalized eigenvalue problem of the form $\mathbf{L}\boldsymbol{v} = \lambda\mathbf{M}\boldsymbol{v}$, where $(\boldsymbol{v}_1^\top\mathbf{L}\boldsymbol{v}_1)/(\boldsymbol{v}_1^\top\mathbf{M}\boldsymbol{v}_1)$ is minimized. Unfortunately, this form did not work well because $\mathbf{L}\boldsymbol{v} = 0$ if $\mathbf{M}\boldsymbol{v} = 0$. Then, it yields many undetermined eigenvalues, and the minimum eigenvector is not always determined with the desired constraint. Therefore, we employed the generalized eigenvalue problem $\mathbf{M}\boldsymbol{v} = \lambda\mathbf{L}_{reg}\boldsymbol{v}$, where $\mathbf{L}_{reg} = \mathbf{L} + \epsilon\mathbf{I}$ is a regularized form of $\mathbf{L}$ to ensure numerical stability.

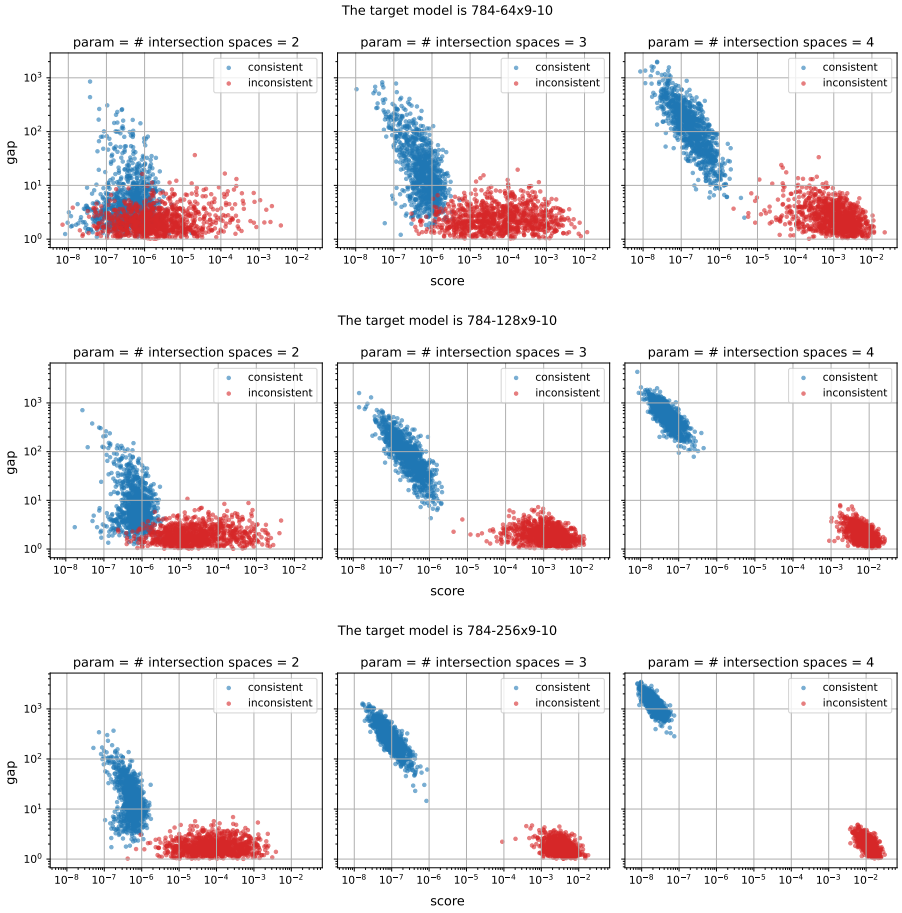


Fig. 8: `IsUnifiedConsistent` for cross-layer extraction on several models, where $\epsilon = 10^{-5}$ for the regularization of $\mathbf{L}$.

λ_1/λ_2, where λ_1 and λ_2 denote the maximum and the second maximum eigenvalues, respectively. The larger the score, the larger $\boldsymbol{v}^\top \mathbf{L} \boldsymbol{v}$ and the smaller $\boldsymbol{v}^\top \mathbf{M} \boldsymbol{v}$, implying that a vector $\boldsymbol{w}_{cross}$ satisfying the desired conditions is unlikely to exist. On the other hand, the larger the gap, the more significantly $\boldsymbol{v}^\top \mathbf{L} \boldsymbol{v}$ is smaller than $\boldsymbol{v}^\top \mathbf{M} \boldsymbol{v}$, implying that there is a high-confidence candidate for $\boldsymbol{w}_{cross}$.

Experimental Verification. We evaluated the unified consistency algorithm experimentally to estimate the number of intersection spaces required to pass the consistency check with high confidence, using models trained on MNIST. We assume the 4th layer contains persistent and dead neurons and performed cross-layer extraction from intersection points in the 5th layer.

Fig. 8 summarizes the experimental results regarding the distinguishability between consistent and inconsistent sets by `IsUnifiedConsistent`. We generated 1000 random consistent and inconsistent sets and applied the unified consistency algorithm. For an inconsistent set, we used a set of consistent intersection spaces, replacing one of them with an inconsistent intersection space.

We observe that three intersection spaces are sufficient to distinguish consistent and inconsistent sets for the models with 128 or 256 hidden units. Even for the model with 64 hidden units, four intersection spaces are sufficient. Our experiment suggests that at most one inconsistent intersection space significantly changes the output of the unified consistency algorithm. Therefore, once a high-confidence consistent set is identified, we can efficiently expand it by adding a new intersection space and checking for consistency.

Time Complexity. A pairwise unified consistency algorithm can distinguish consistent and inconsistent pairs to a limited extent, but it lacks sufficient distinguishability when the number of hidden units is small. We recommend first identifying a high-confidence pair or triple, and then expand the set one by one. When we initially search over triples, the expected complexity is $O(n^3)$.

The unified consistency algorithm is slightly more complex than the conventional one because it solves a generalized eigenvalue problem instead of a standard one. Their theoretical complexities are equivalent, though the generalized problem is slower by a constant factor for matrices of the same size.

5.2 Signature/Span Recovery

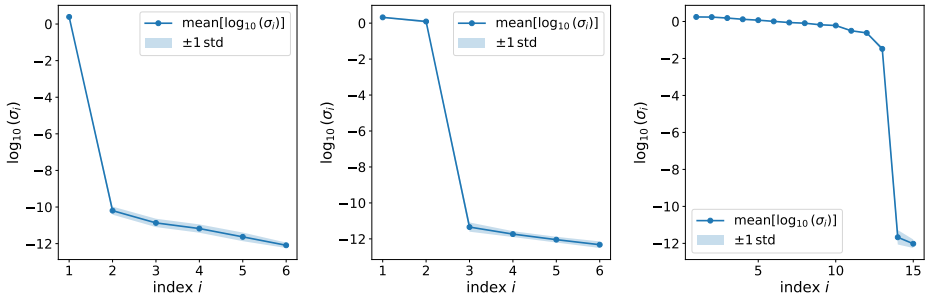
Given a set of consistent intersection spaces, we next recover the signature, and the same Algorithm 3 can be used for this purpose. Algorithm 3 returns the maximum eigenvector that can be a candidate of $\boldsymbol{w}_{cross}$. When we use a sufficient number of intersection spaces, the rank of the matrix $\mathbf{L}$ is one less than the rank of the matrix $\mathbf{M}$. Thus, the maximum eigenvector is the only vector satisfying $\boldsymbol{v}^\top \mathbf{L} = \mathbf{0}$ and $\boldsymbol{v}^\top \mathbf{M} \neq \mathbf{0}$, that is $\boldsymbol{w}_{cross}$.

Single Persistent Neuron, $|\mathcal{P}| = 1$, and Signature Recovery. We discuss the simplest case, i.e., only one persistent neuron in the ℓth layer. Then, the maximum eigenvector is

$$\boldsymbol{w}_{cross} = \begin{pmatrix} \boldsymbol{w}^{(\ell+1)}_{k,\{1,\ldots,p-1,p+1,\ldots,d_{\ell-1}\}} \\ w^{(\ell+1)}_{k,p} \boldsymbol{w}^{(\ell)}_p \end{pmatrix},$$

which directly contains the signature of the persistent neuron, $\boldsymbol{w}^{(\ell)}_p$.

We can successfully recover the signature, but we cannot recover the corresponding sign and bias. More precisely, we do not need to recover them because they do not affect the model's output as long as the neuron remains persistent. Therefore, persistent neurons imply that ReLU is skipped. In other words, the recovered model does not have the ReLU function in persistent neurons. The sign and bias are linearly expanded into the next layer, and we can extract them together with the next-layer extraction.



(a) 784-64×9-10 (b) 784-128×9-10 (c) 784-256×9-10

Fig. 9: Scree plot of singular values, where $\epsilon = 10^{-13}$ for the regularization of $\mathbf{L}$.

Multiple Persistent Neurons and Span Recovery. When multiple persistent neurons exist, an extra analysis is required. The maximum eigenvector contains

$$\sum_{j \in \mathcal{P}} w_{k,j}^{(\ell+1)} \boldsymbol{w}_j^{(\ell)},$$

that is a linear combination of the weights of persistent neurons. We repeat the cross-layer extraction against at least $|\mathcal{P}|$ target neurons in the $(\ell+1)$th layer and obtain $|\mathcal{P}|$ bases, $\tilde{\boldsymbol{w}}_1^{(\ell)}, \tilde{\boldsymbol{w}}_2^{(\ell)}, \ldots, \tilde{\boldsymbol{w}}_{|\mathcal{P}|}^{(\ell)}$. The span of these obtained bases is the same as the span of vectors $\boldsymbol{w}_j^{(\ell)}$ for $j \in \mathcal{P}$.

We cannot recover the original basis because we cannot know $w_{k,j}^{(\ell+1)}$ as long as these neurons are persistent. Therefore, we use the basis $\tilde{\boldsymbol{w}}_1^{(\ell)}, \tilde{\boldsymbol{w}}_2^{(\ell)}, \ldots, \tilde{\boldsymbol{w}}_{|\mathcal{P}|}^{(\ell)}$ as the "weights" of the corresponding persistent neurons. The ReLU of these neurons must be removed in the recovered model because they were originally persistent and never suppressed by the ReLU. Similar to the case of $|\mathcal{P}| = 1$, we do not need to consider sign and bias because there are no ReLU functions.

Accuracy of Model Extraction. Model extraction via cross-layer extraction can, in principle, recover the target model exactly, as long as neurons classified as persistent or dead indeed remain in those respective states. If a neuron classified as persistent were to become inactive, or a neuron classified as dead were to become active, the extracted model would no longer be guaranteed to produce identical outputs to the original model. On the other hand, the neurons we classify as persistent or dead are those whose activation states never switched between active and inactive, even after an exponential number of queries. Therefore, the extracted model reproduces the outputs of the original model with overwhelming probability.

Experimental Verification. To verify the feasibility of cross-layer extraction, we conducted experiments using the same model as in the unified consistency

algorithm. We consider models in which the 4th layer contains 1, 2, and 13 persistent neurons for 64, 128, and 256 hidden units, respectively, and attempt to recover them via cross-layer extraction from the intersection spaces in the 5th layer. For the models with 64 and 128 hidden units, we collected 6 vectors via cross-layer extraction, while for the model with 256 hidden units, we collected 15 vectors. For each hidden-unit configuration, we repeated the experiment 30 times. Fig. 9 shows the scree plots of the corresponding singular values.

From these experiments, we observed a clear rank deficiency, and the observed rank corresponds to the number of persistent neurons. We also evaluated the principal angles between the space spanned by the recovered vectors and that spanned by the true (persistent) neuron weights. The largest principal angle was approximately 10^{-8}, which indicates successful recovery of the original span.

5.3 Discussion

Avoiding Local Persistent Neurons. When using intersection spaces $\mathcal{S}_i$, each coordinate of $f^{(\ell-1)}(\mathcal{S}_i)$ should attain both active and inactive values, unless it corresponds to a persistent or dead neuron. If a neuron is locally dead, its coefficient cannot be recovered. If it is locally persistent, its weight may be absorbed into the linear combination of persistent neurons. Once this happens, removing its influence is difficult because its weight is not necessarily orthogonal to the persistent neurons.

Therefore, we need $|\mathcal{P}|$ target neurons that are free of local persistence. This is expected to be feasible, since their signatures and signs were successfully extracted in the preceding-layer attack, implying that they are neither persistent nor dead. If we still cannot collect enough such neurons, the best strategy is to treat them as persistent; the recovered model then remains correct as long as local persistence holds for the input.

Recovering Future Neurons. Cross-layer extraction recovers the signature or span of persistent neurons. Once these are recovered, we assign appropriate weights and remove the ReLU, completing the attack up to the ℓth layer. From the next layer onward, we return to the original extraction procedure.

As long as the classification of persistent/dead neurons is consistent, the subsequent layers pose no additional difficulty. If no persistent or dead neurons appear in the next layer, we fully revert to the original procedure. More commonly, when persistent or dead neurons are present, we (i) recover signatures of all recoverable neurons, (ii) recover their signs, and (iii) apply cross-layer extraction to obtain the signature/span of persistent neurons.

The recovered model is not strictly identical to the original model. The attacker exploits only inputs on which persistent/dead neurons exhibit the assumed behavior; thus, if an input induces contradictory activation patterns, the recovered model may produce different outputs.

On the Soft-Label Setting. Our main focus is the hard-label setting, but attacks on the soft-label setting have also been studied [5,17,7,15]. We believe that cross-layer extraction or its extension is also helpful in the soft-label setting. In the soft-label setting, one can compute second-order derivatives of the output logits, which directly reveal activation boundaries (without going through the decision boundary). However, similar to the hard-label setting, activation boundaries associated with persistent neurons are inaccessible from the input, which causes inevitable errors in the recovered forward local linear map. Similar to the hard-label setting, the soft-label attacker also must peel off already recovered layer to attack the subsequent layer. Here, since the only available recovered map contains errors, we face the same issue. Since the soft-label attacker is strictly more powerful than the hard-label attacker, it is reasonable to expect that techniques analogous to cross-layer extraction could also be applied to address this issue in the soft-label setting. We leave the development of a methodology specialized for the soft-label setting as an open question.

6 Conclusion

In this paper, we revisited the hard-label cryptanalytic model extraction proposed at Eurocrypt 2025 [4]. Carlini et al. claimed that the attack runs in polynomial time under the assumption that all neurons switch between active and inactive states with equal probability. However, we showed that this assumption is unlikely to hold in practice. In particular, the existence of persistent (always active) neurons introduces non-negligible and unavoidable noise, which can render the attack infeasible. To address this issue, we proposed *cross-layer extraction*, which recovers a model that behaves identically to the original as long as persistent and dead neurons remain active and inactive, respectively. If there are n ε-persistent neurons and m ε-dead neurons, then by the union bound, the recovered model is correct with probability at least $1 - (n + m)\varepsilon$.

Finally, we emphasize that both our attack and the original attack are entirely theoretical, assuming that intersection points can be extracted without noise whenever they exist. In practice, however, unavoidable numerical errors, such as machine epsilon, tend to accumulate as deeper layers are analyzed. Therefore, from an engineering perspective, it remains an open problem whether the attacker can reliably extract these models under realistic noise conditions.

Acknowledgments. This work was supported by JST K Program Grant Number JPMJKP24C1, Japan.

References

1. Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology - ASIACRYPT

2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. LNCS, vol. 8873, pp. 63–84. Springer (2014). https://doi.org/10.1007/978-3-662-45611-8_4
2. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. J. Cryptol. **23**(4), 505–518 (2010). https://doi.org/10.1007/S00145-010-9062-1
3. Canales-Martínez, I.A., Santos, D.: Extracting some layers of deep neural networks in the hard-label setting. In: Escudero, D., Damgård, I. (eds.) Progress in Cryptology - LATINCRYPT 2025 - 9th International Conference on Cryptology and Information Security in Latin America, Medellín, Colombia, October 1-3, 2025, Proceedings. pp. 399–421. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-3-032-06754-8_15, https://doi.org/10.1007/978-3-032-06754-8_15
4. 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. In: Fehr, S., Fouque, P. (eds.) Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part I. LNCS, vol. 15601, pp. 364–396. Springer (2025). https://doi.org/10.1007/978-3-031-91107-1_13
5. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. LNCS, vol. 12172, pp. 189–218. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_7
6. Chen, Y., Dong, X., Guo, J., Shen, Y., Wang, A., Wang, X.: Hard-label cryptanalytic extraction of neural network models. In: Chung, K., Sasaki, Y. (eds.) Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VIII. LNCS, vol. 15491, pp. 207–236. Springer (2024). https://doi.org/10.1007/978-981-96-0944-4_7
7. Chen, Y., Dong, X., Ma, R., Shen, Y., Wang, A., Yu, H., Wang, X.: Delving into cryptanalytic extraction of prelu neural networks. In: Hanaoka, G., Yang, B. (eds.) Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part II. LNCS, vol. 16246, pp. 576–607. Springer (2025). https://doi.org/10.1007/978-981-95-5096-8_18
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002). https://doi.org/10.1007/978-3-662-04722-4
9. 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
10. Foerster, H., Mullins, R.D., Shumailov, I., Hayes, J.: Beyond slow signs in high-fidelity model extraction. In: Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J.M., Zhang, C. (eds.) Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December

10 - 15, 2024 (2024), http://papers.nips.cc/paper_files/paper/2024/hash/22ae669a35bb9e70eb93ab77c1eff5b4-Abstract-Conference.html
11. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Gordon, G.J., Dunson, D.B., Dudík, M. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011. JMLR Proceedings, vol. 15, pp. 315–323. JMLR.org (2011), http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf
12. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. pp. 1026–1034. IEEE Computer Society (2015). https://doi.org/10.1109/ICCV.2015.123
13. 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.) 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020. pp. 1345–1362. USENIX Association (2020), https://www.usenix.org/conference/usenixsecurity20/presentation/jagielski
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
15. Liu, H., Siproudhis, A., Experton, S., Lorenz, P., Boura, C., Peyrin, T.: Navigating the deep: End-to-end extraction on deep neural networks. In: Daemen, J., Thomé, E. (eds.) Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part VI. pp. 482–512. Lecture Notes in Computer Science, Springer (2026). https://doi.org/10.1007/978-3-032-25333-0_17
16. 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
17. Martinez, I.A.C., 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.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III. LNCS, vol. 14653, pp. 3–33. Springer (2024). https://doi.org/10.1007/978-3-031-58734-4_1
18. Milli, S., Schmidt, L., Dragan, A.D., Hardt, M.: Model reconstruction from model explanations. In: danah boyd, Morgenstern, J.H. (eds.) Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019. pp. 1–9. ACM (2019). https://doi.org/10.1145/3287560.3287562
19. Miura, T., Shibahara, T., Yanai, N.: MEGEX: data-free model extraction attack against gradient-based explainable AI. In: Proceedings of the 2nd ACM Workshop on Secure and Trustworthy Deep Learning Systems, SecTL 2024, Singapore, July 2, 2024. pp. 56–66. ACM (2024). https://doi.org/10.1145/3665451.3665533
20. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A.W., Kavukcuoglu, K.: Wavenet: A generative model

for raw audio. In: Black, A.W. (ed.) The 9th ISCA Speech Synthesis Workshop, SSW 2016, Sunnyvale, CA, USA, September 13-15, 2016. p. 125. ISCA (2016), https://www.isca-archive.org/ssw_2016/vandenoord16_ssw.html

21. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 139, pp. 8748–8763. PMLR (2021), http://proceedings.mlr.press/v139/radford21a.html
22. Rolnick, D., Kording, K.P.: Reverse-engineering deep relu networks. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event. Proceedings of Machine Learning Research, vol. 119, pp. 8178–8187. PMLR (2020), http://proceedings.mlr.press/v119/rolnick20a.html
23. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 601–618. USENIX Association (2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 5998–6008 (2017), https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

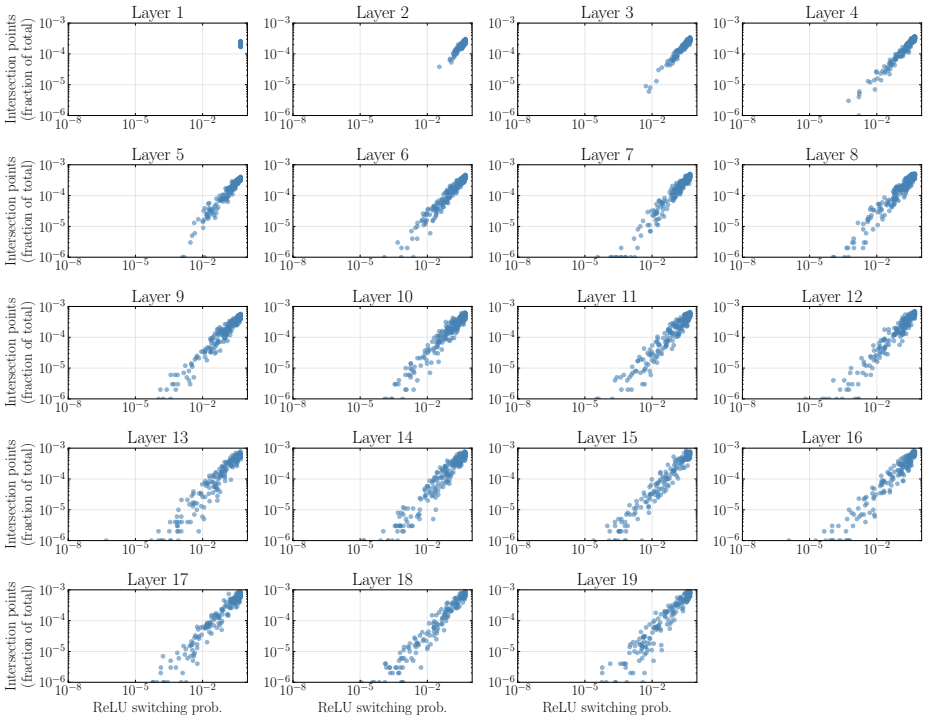


Fig. 10: Switching probability versus the proportion of discovered intersection points in all layers of an untrained MLP.

A Additional Experimental Results

A.1 Switching Probability and Proportion of Intersection Points

Figs. 10 to 12 show the relationship between the switching probability of each neuron and the number of discovered intersection points across all layers. Each point corresponds to a neuron, where the vertical axis indicates the number of discovered intersection points normalized by the total number of explored points, and the horizontal axis represents the switching probability of each neuron. Under all experimental conditions, a strong correlation between the two quantities can be observed.

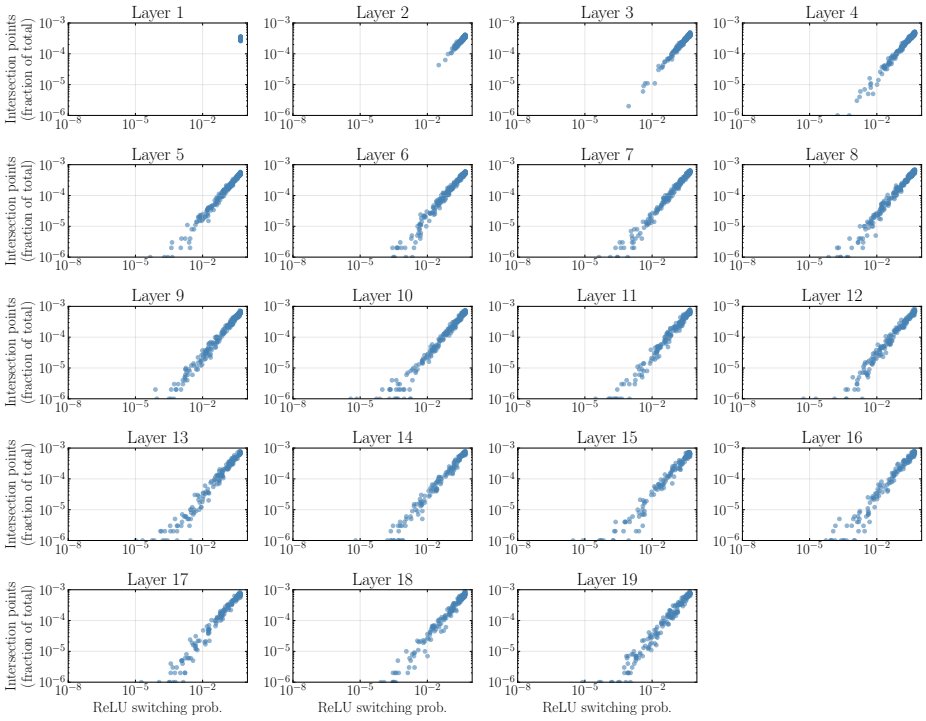


Fig. 11: Switching probability versus the proportion of discovered intersection points in all layers of an MLP trained on MNIST.

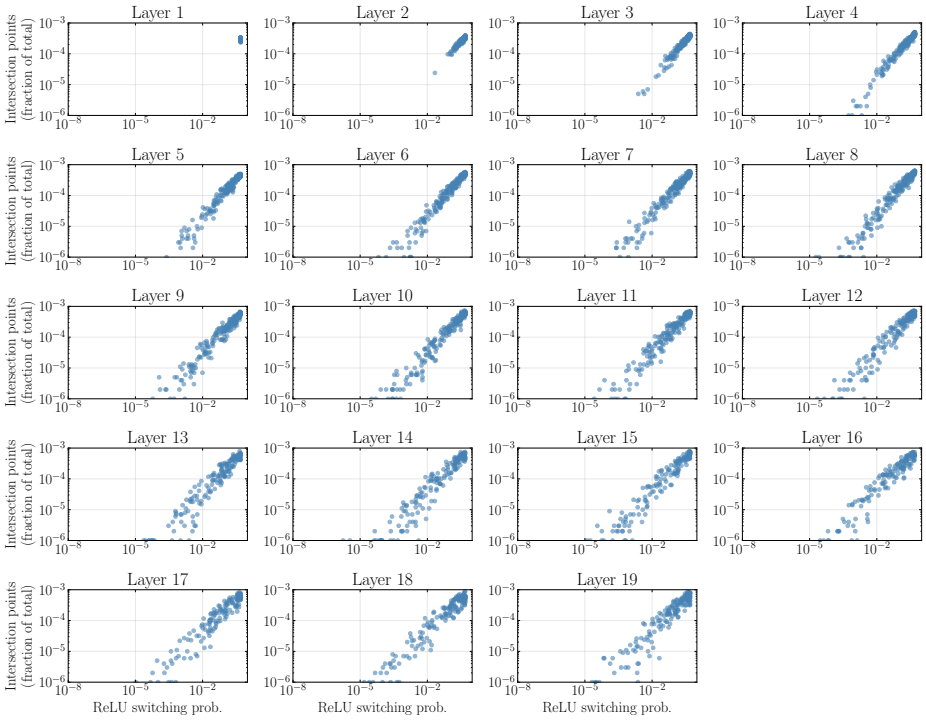


Fig. 12: Switching probability versus the proportion of discovered intersection points in all layers of an MLP trained on FMNIST.