Breaking Slope and Structure Restrictions: Broadening Hard-Label Cryptanalytic Extraction of PReLU Neural Networks

Ruijie Ma[1], Yi Chen[2(✉)], Jiarui Zhang[3], Hongbo Yu[1,4,5], and Xiaoyun Wang[2,3,4,5,6]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China
marj21@mails.tsinghua.edu.cn, yuhongbo@tsinghua.edu.cn
[2] Institute for Advanced Study, Tsinghua University, Beijing, China
chenyi2023@tsinghua.edu.cn
[3] School of Cryptographic Science and Engineering, Shandong University, Jinan, China
202417051@mail.sdu.edu.cn
[4] State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University, Beijing, China
xiaoyunwang@tsinghua.edu.cn
[5] Zhongguancun Laboratory, Beijing, China
[6] Shandong Institute of Blockchain, Shandong, China

Abstract. This paper studies the problem of model parameter extraction of PReLU neural networks in the hard-label setting, the most challenging setting. Existing attacks on PReLU neural networks suffer from two fundamental restrictions: (1) the learnable slopes in PReLU activations are restricted to smaller than 1, not conforming to the standard definitions of PReLU activations; (2) they do not apply to expansive PReLU neural networks. In this paper, for the first time, we break the two restrictions by proposing a new attack in the hard-label setting.

Our breakthroughs stem from two new techniques and an important finding. First, we propose a new network isomorphism, called flip-and-scaling, which helps break the slope restriction and build a new extraction framework. Second, we find that there are linear constraints on the internal states of expansive PReLU neural networks, and give the exact number of linear constraints. Third, we propose a new neuron signature recovery method for expansive PReLU neural networks, which overcomes the challenge brought by linear constraints and breaks the structure restriction. The correctness and effectiveness of our work have been fully verified by experiments on several hundred expansive PReLU neural networks. Overall, our work not only overcomes the restrictions of existing attacks but also provides some inspiration for future work.

Keywords: Model Parameter Extraction · Cryptanalysis · Expansive PReLU Neural Networks · Network Isomorphism · Hard-Label Setting.

1 Introduction

Model Parameter Extraction in The Black-box Scenario. The problem of extracting model parameters of neural networks has been studied since 1990s [1–3, 21], and has gained increasing attention in recent years [4–10, 12–15, 17–19, 22–24].

Functionally Equivalent Extraction. The most stringent target in model parameter extraction is functionally equivalent extraction [13], which seeks to recover a parameter set $\widehat{\theta}$ such that $f_{\widehat{\theta}}(X) = f_\theta(X)$ holds identically for all inputs X, where f is the neural network architecture and θ represents the original model parameters. The model $f_{\widehat{\theta}}$ is called the functionally equivalent model of f_θ.

The Raw Output and Hard-Label Settings. Functionally equivalent extraction usually assumes that the adversary only has access to the feedback of the network [6], which could be the raw output $f_\theta(X)$, hard-label, or otherwise. In the raw output setting (respectively, hard-label setting), the feedback is the raw output (respectively, hard-label). The hard-label setting, the most challenging setting, has been long considered a strong defense against functionally equivalent extraction [6, 13, 22].

Network Isomorphism. The recovered model parameters $\widehat{\theta}$ could differ from the original parameters θ, which is attributed to network isomorphism, one of the most fundamental properties of neural networks. Specifically, for some types of neural networks, some special transformations applied to model parameters do not affect the raw output. Such a property is the basis of model parameter extraction attacks [8, 22].

Cryptanalytic Extraction Framework For ReLU Neural Networks. At Crypto 2020, Carlini et al. propose the first cryptanalytic extraction attack that theoretically applies to ReLU neural networks (i.e., ReLU activation-based fully connected neural networks) with an unlimited number of hidden layers [6], formally opening the research direction of cryptanalytic model extraction.

Carlini et al. propose a cryptanalytic extraction framework for ReLU neural networks [6]. This attack framework recovers model parameters in a layer-wise and neuron-wise manner. Concretely, the adversary recovers model parameters starting from layer 1 to the last layer. In each layer, the model parameters are recovered neuron by neuron. The parameter recovery of each neuron is composed of two steps below:

- **Neuron Signature Recovery:** Denote by $\widehat{A}_i^k$ the final recovered weight of the i-th neuron in layer k. The neuron signature recovery aims to recover the neuron signature $\tilde{A}_i^k \in \{\widehat{A}_i^k, -\widehat{A}_i^k\}$.
- **Neuron Sign Recovery:** Determine the correct sign of signature, i.e., checking whether $\tilde{A}_i^k = \widehat{A}_i^k$ or $\tilde{A}_i^k = -\widehat{A}_i^k$.

Based on the cryptanalytic extraction framework above, researchers have made significant breakthroughs on ReLU neural networks. The attack in [6] requires access to the raw output of neural networks, and its time complexity is exponential for expansive networks. At Eurocrypt 2024, Canales Martinez et al. propose new neuron sign recovery techniques, reducing the time complexity to polynomial [18]. At Asiacrypt 2024, Chen et al. propose the first attack in the hard-label setting [7], which requires exponential time complexity. At Eurocrypt 2025, Carlini et al. propose new signature and sign recovery methods from the geometric perspective, reducing the time complexity in the hard-label setting to polynomial [5].

Restrictions of Existing Attacks on PReLU Neural Networks. The above extraction framework, initially designed for ReLU neural networks, has also been applied to fully connected networks with PReLU activation functions [8, 20], referred to as PReLU neural networks. At Asiacrypt 2025, Chen et al. propose the first cryptanalytic extraction attack against PReLU neural networks in the raw output setting [8]. Recently, Qi et al. have applied the methods [5], proposed at Eurocrypt 2025 for ReLU neural networks, to PReLU neural networks in the hard-label setting.

These existing attacks [8,20] on PReLU neural networks both face two strong restrictions as follows:

- **Slope Restriction:** All slopes are assumed to be greater than 0 and smaller than 1 simultaneously. However, this constraint is overly restrictive and inconsistent with the standard definition of the PReLU activation function [11], which merely requires that all slopes be greater than 0.
- **Structure Restriction:** Existing attacks [8,20] do not apply to expansive networks. Concretely, existing attacks [8,20] only work when the number of neurons in each hidden layer does not exceed that in previous hidden layers.

The slope restriction is attributed to the neuron sign recovery in [8, 20]. Concretely, consider the i-th neuron in layer k, denoted by η. After neuron signature recovery of η, the attacks in [8, 20] further recovers a pair of weights (w_+, w_-), where $(w_+, w_-) = (a, s_i^k a)$ when $\tilde{A}_i^k = \widehat{A}_i^k$, or $(w_+, w_-) = (s_i^k a, a)$ when $\tilde{A}_i^k = -\widehat{A}_i^k$. Here, s_i^k is the slope corresponding to η. Under the restriction that $0 < s_i^k < 1$, one knows that the neuron sign is 1 (i.e., $\tilde{A}_i^k = \widehat{A}_i^k$) if $|w_+| > |w_-|$, or -1 if $|w_-| > |w_+|$. Without the restriction $0 < s_i^k < 1$, existing attacks [8,20] can not recover the neuron sign.

The structural constraint is attributed to the slope recovery in [20] and the neuron signature recovery in [8]. In [20], Qi et al claim that the hard-label-based signature recovery method proposed in [5] applies to PReLU neural networks, and the only obstacle is the recovery of slopes. The slope recovery method in [20] requires that the inverse of A^k exists, leading to the structure restriction $d_k \leqslant d_{k-1}$, i.e., the network is non-expansive. However, we find in this paper that the hard-label-based signature recovery method in [5] actually does not apply to expansive PReLU neural networks. As for the neuron signature recovery method

in [8], it relies on solving a linear system of equations. To make the system solvable, the network must be non-expansive [8].

Our Targets in This Paper. In this paper, the primary target is to break the slope and structure restrictions in existing cryptanalytic extraction attacks against PReLU neural networks.

To break the slope restriction, there are two possible directions. In the first direction, the existing cryptanalytic extraction framework, originally designed for ReLU-based neural networks, is still applied unchanged. In this case, the research emphasis shifts to developing techniques capable of recovering the neuron sign and slope without extra assumptions. The second direction is to reconsider the necessity of recovering neuron signs. If one could recover a functionally equivalent model even if the neuron sign is -1, it would be unnecessary to apply an extra slope restriction.

In this paper, we follow the second direction, starting by studying fundamental properties of PReLU neural networks, such as the network isomorphisms that directly determine the necessity of recovering neuron signs. There are more emerging works on new types of neural networks [20, 23] and researchers have been using the cryptanalytic extraction framework directly, originally designed for ReLU-based neural networks, without making any changes to it. If we successfully break the slope restriction in the second direction, it would be a good example to show that it is better to delve into the fundamental mathematical properties of new neural networks.

Regarding the structure restriction, we also try to figure out why the method [5] proposed at Eurocrypt 2025 does not work for expansive PReLU neural networks, the secondary target of this paper.

1.1 Our Contributions

Results. All the preset targets have been achieved in this paper:

- **Break Slope and Structure Restrictions.** We propose a new model parameter recovery attack on PReLU neural networks in the hard-label setting, which simultaneously breaks both the slope and structure restrictions. In other words, our attack applies to expansive PReLU neural networks and imposes no restrictions on the neuron slopes. Moreover, the step of neuron sign recovery is removed in our attack, since we demonstrate that it is sufficient to obtain a functionally equivalent model of PReLU neural networks, even if the neuron signs are not recovered.
- **Explain Why the Neuron Signature Recovery Method [5] Fails.** We present a clear explanation of why the neuron signature recovery method [5] (designed for ReLU neural networks) fails for expansive PReLU neural networks. At a high level, the failure is caused by linear constraints on the internal states of expansive PReLU neural networks. And the number of linear constraints is determined by the network structure.

Techniques and Findings. Our attack is based on two new proposed techniques and one important finding.

- **New Network Isomorphism in PReLU Neural Networks.** The first technique is a new network isomorphism, named flip-and-scaling, that applies to neural networks with PReLU activation functions. The flip-and-scaling isomorphism ensures that the recovered model is still a functionally equivalent model when the neuron signature is $\widetilde{A}_i^k = -\widehat{A}_i^k$. It is based on the finding that the scaling factors applied to weights and bias of neurons can be negative if we replace the initial slope s_j^k with the reciprocal $\frac{1}{s_j^k}$. More details are introduced in Section 4.
- **Linear Constraints in Expansive PReLU Neural Networks.** The important finding is that the preactivation input of layer k is subject to $d_k - \text{rank}(A^k)$ linear constraints, where d_k is the number of neurons in layer k, $\text{rank}(A^k) \in \mathbb{R}^{d_k \times d_{k-1}}$ is the rank of the weight matrix A^k of layer k, and d_{k-1} is the number of neurons in layer $k-1$. If $d_k > d_{k-1}$, meaning that layer k is expansive, the inequation $d_k - \text{rank}(A^k) > 0$ holds. After applying the neuron splitting technique [8] to layer k, the $d_k - \text{rank}(A^k)$ linear constraints will be induced on the output of layer k, which prevents the extension of the neuron signature recovery method [5] (designed for ReLU neural networks) to expansive PReLU neural networks. Concrete analysis is presented in Section 5.
- **New Signature Recovery Method.** We propose a joint neuron signature recovery method for expansive PReLU networks. The key insight is that although additional linear constraints (analyzed in Section 5) prevent independent recovery of each neuron, the neuron slopes of layer k couple all neurons in layer $k+1$. Our method reconstructs a set of affine spaces (each corresponding to a neuron in layer $k+1$) and solves a nonlinear system that simultaneously recovers the extended neuron signatures of layer $k+1$ and the neuron slopes of layer k. Two improvements, the residual analysis and a progressive decoupling strategy, are further proposed to improve the attack.

We have performed numerous experiments to verify the correctness and effectiveness of our techniques. Specifically, we have collected several hundred expansive PReLU neural networks, including random networks and the real networks trained on the MNIST dataset, to verify the effectiveness of the new signature recovery method (refer to Section 7). Besides, some support experiments are also performed to verify the new network isomorphism (see Section 4), and the analysis of the number of linear constraints (see Section 5). The implementation of all the experiments, as well as the networks tested in this paper, is available at https://xxxxxx.

As introduced, there is a growing trend in which researchers directly apply existing cryptanalytic extraction frameworks to other unexplored neural networks. We hope our work will positively influence future research on model parameter extraction, encouraging greater attention to the fundamental mathematical properties underlying different neural network architectures.

Organization. Section 3 gives an overview of our attack. Section 4 introduces the flip-and-scaling isomorphism that eliminates the need for sign recovery. Section 5 analyzes the linear constraints arising in expansive PReLU networks. Section 6 presents our joint neuron signature recovery method. Section 7 reports experimental results.

2 Preliminaries

2.1 Basic Definitions and Notations

Definition 1 (n-Deep Neural Network). *An n-deep neural network f_θ is a function parameterized by θ that maps input x in input space $\mathcal{X}$ into output y in output space $\mathcal{Y}$. The function $f : \mathcal{X} \to \mathcal{Y}$ is composed of alternating linear layers f_i and non-linear layers σ_i, that is:*

$$f_\theta = f_{n+1} \circ \sigma_n \circ f_n \cdots \circ \sigma_2 \circ f_2 \circ \sigma_1 \circ f_1. \tag{1}$$

Definition 2 (Fully Connected Layer). *The k-th linear layer is a fully connected layer which is a function $f_k : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ given by an affine transformation*

$$f_k(X) = A^k X + B^k, \tag{2}$$

where $A^k \in \mathbb{R}^{d_k \times d_{k-1}}$ is the weight matrix and $B^k \in \mathbb{R}^{d_k}$ is the bias vector. Denote

$$A^k = \begin{bmatrix} a^k_{1,1} & a^k_{1,2} & \cdots & a^k_{1,d_{k-1}} \\ a^k_{2,1} & a^k_{2,2} & \cdots & a^k_{2,d_{k-1}} \\ \vdots & \vdots & \ddots & \vdots \\ a^k_{d_k,1} & a^k_{d_k,2} & \cdots & a^k_{d_k,d_{k-1}} \end{bmatrix}, B^k = \begin{bmatrix} b^k_1 \\ b^k_2 \\ \vdots \\ b^k_{d_k} \end{bmatrix}. \tag{3}$$

In the rest of this paper,the i-th row of A^k is denoted by A^k_i, and the i-th column of A^k is denoted by $A^k_{:,i}$.

Definition 3 (Activation Layer). *The k-th non-linear layer σ_k is an activation layer composed of d_k non-linear activation functions $\sigma_{k,i}$ for $i \in \{1, 2, \cdots, d_k\}$.*

This paper focuses on the PReLU activation function [11] that is defined as

$$\sigma_{k,i}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ s^k_i \times x, & \text{if } x < 0 \end{cases} \tag{4}$$

where $s^k_i > 0$ is the slope determined by the training of neural networks. We denote the preactivation input of σ_k in layer k by $Y^k = \left[y^k_1, y^k_2, \cdots, y^k_{d_k}\right]^\top$. Then, σ_k can be expressed as

$$\sigma_k(Y^k) = I^k Y^k,\ I^k = \begin{bmatrix} \tau^k_1 & 0 & \cdots & 0 \\ 0 & \tau^k_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \tau^k_{d_k} \end{bmatrix}, \tau^k_i = \begin{cases} 1, & \text{if } y^k_i \geq 0 \\ s^k_i, & \text{if } y^k_i < 0 \end{cases} \tag{5}$$

Definition 4 (Neuron). *A neuron is a function determined by the corresponding weight matrix, bias vector, and activation function. Formally, the i-th neuron of layer k is the function* η_i^k *defined by*

$$\eta_i^k(Z^{k-1}) = \sigma_{k,i}(A_i^k Z^{k-1} + b_i^k) \tag{6}$$

where Z^{k-1}*, the output of layer* $k-1$*, is defined as:*

$$Z^{k-1} = \begin{cases} X, & \text{if } k = 1 \\ \sigma_{k-1}(Y^{k-1}), & \text{if } k > 1 \end{cases}, \tag{7}$$

where X *is the input of the whole network. There are* d_k *neurons in layer* k*.*

For neuron η_i^k, its incoming weight is A_i^k and the outgoing weight is $A_{:,i}^{k+1}$.

Definition 5 (Network Architecture). *The architecture of a fully connected neural network captures the structure of* f_θ*: (a) the number of layers, (b) the dimension* d_k *of each layer* $k \in \{0, 1, \cdots, n+1\}$*. We say that* d_0 *and* d_{n+1} *are the input and output dimensions of the neural network. The* n*-deep neural network* f_θ *is denoted by* d_0*-*d_1*-*$\cdots$*-*d_{n+1}*.*

Definition 6 (Model Parameter). *The parameters* θ *of an* n*-deep neural network* f_θ *are the concrete assignments to the weights* A^k*, biases* b^k *for* $k \in \{1, 2, \cdots, n+1\}$*, and slopes* $S^k = [s_1^k, s_2^k, \cdots, s_{d_k}^k]$ *for* $k \in \{1, 2, \cdots, n\}$*.*

Definition 7 (Hard Label). *Let* $f_\theta : \mathcal{X} \to \mathcal{Y}$ *be an* n*-deep neural network with* $\mathcal{Y} = \mathbb{R}^{d_{n+1}}$*. The hard label* z *on the outputs* $f_\theta(X)$ *is the coordinate of the maximum of* $f_\theta(X)$*. If multiple entries in* $f_\theta(X)$ *have the same maximum value, the hard-label* z *is the smallest coordinate of these equal entries.*

Definition 8 (Dual Point). *Let* $\mathcal{V}(\eta; X)$ *denote the value that neuron* η *takes with input* $X \in \mathbb{R}^{d_0}$ *before applying its corresponding activation function* σ*. A dual point is an input* $X \in \mathcal{X}$ *that is both on the decision boundary and a critical hyperplane. Specifically, a dual point* X *satisfies* $\exists\, \eta, \mathcal{V}(\eta; X) = 0$*, and makes at least two entries of the output* $f_\theta(X)$ *have the same maximum value.*

2.2 Neuron Signature Recovery in the Hard-Label Setting [5]

At Eurocrypt 2025, Carlini et al. proposed a neuron signature recovery method for ReLU neural networks, which requires only a polynomial number of queries and a polynomial time complexity in the hard-label setting [5].

The core idea of the neuron signature recovery method in [5] is as follows. Consider the recovery of the i-th neuron η_i^{k+1} in layer $k+1$. By collecting dual points, the adversary first reconstructs the $(d_k - 1)$-dimensional affine space V corresponding to η_i^{k+1}, where $V = \{Z^k | A_i^{k+1} Z^k + b_i^{k+1} = 0\}$. Then, compute its orthogonal complement space $V^\perp = \text{span}(A_i^{k+1})$, and extract $\tilde{A}_i^{k+1}$ through the basis of $V^\perp$.

The affine space V is reconstructed by collecting dual points corresponding to the neuron η_i^{k+1}. The methods of finding dual points, filtering dual points, and reconstructing the affine space, introduced in [5], are directly adopted in this paper, and not repeated here. For more details, refer to [5].

2.3 Neuron Splitting Technique For PReLU Neural Networks [8]

At Asiacrypt 2025, Chen et al. propose a neuron splitting technique for PReLU neural networks [8]. This technique is adopted in existing attacks [8, 20] on PReLU neural networks, and will be adopted in our attack.

The neuron splitting technique is based on a property as follows. Consider the j-th neuron η_j^k in layer k. The PReLU activation function corresponding to η_j^k can be reformulated as

$$\mathrm{PReLU}(y_j^k) = \max(y_j^k, 0) + s_j^k \times \min(y_j^k, 0), \tag{8}$$

where y_j^k is the preactivation input of η_j^k, and s_j^k is the slope.

Based on Eq.(8), the neuron splitting technique replaces η_j^k with a pair of neurons (denoted by $\overline{\eta}_j^k$ and $\overline{\eta}_{j+d_k}^k$) that share the same incoming weights and bias. Besides, for the neuron $\overline{\eta}_j^k$ (respectively, $\overline{\eta}_{j+d_k}^k$), its activation function is set to $\max(y_j^k, 0)$ (respectively, $\min(y_j^k, 0)$), and its outgoing weights are set to $A_{:,j}^{k+1}$ (respectively, $s_j^k \cdot A_{:,j}^{k+1}$). Fig. 1 shows the system diagram of the neuron splitting technique. For convenience, we call the layer composed of the $2d_k$ neurons ($\overline{\eta}_j^k, j \in \{1, \cdots, 2d_k\}$) the extended layer k. The output of the extended layer k is denoted by $\overline{Z}^k$ where

$$\begin{aligned} &\overline{Z}^k = \left[\overline{z}_1^k, \overline{z}_2^k, \cdots, \overline{z}_{2d_k}^k\right]^\top, \\ &\overline{z}_j^k = \max(y_j^k, 0) \text{ and } \overline{z}_{j+d_k}^k = \min(y_j^k, 0), j \in \{1, \cdots, d_k\}. \end{aligned} \tag{9}$$

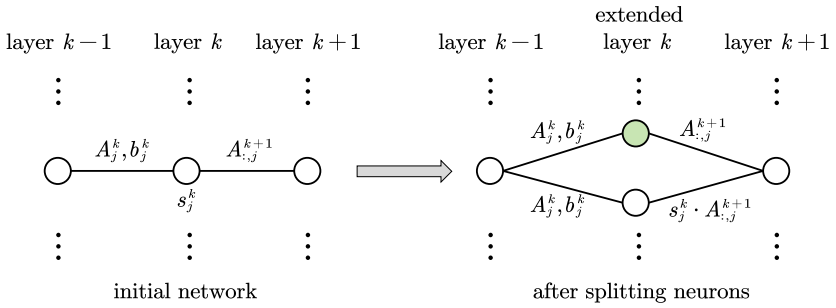

Fig. 1. System diagram of the neuron splitting technique [8]. Each neuron in layer k is replaced by a pair of neurons $\overline{\eta}_j^k$ (the solid circle in the extended layer k) and $\overline{\eta}_{j+d_k}^k$ (the hollow circle), which share the same incoming weights A_j^k and biases b_j^k, leading to the same preactivation input y_j^k. The outgoing weights are $A_{:,j}^{k+1}$ and $s_j^k \cdot A_{:,j}^{k+1}$ respectively. The activation functions are $\max(y_j^k, 0)$ and $\min(y_j^k, 0)$ respectively.

Now, consider the i-th neuron η_i^{k+1} in layer $k+1$. Its extended incoming weights are denoted by $\overline{A}_i^{k+1}$ where

$$\begin{aligned}\overline{A}_i^{k+1} &= \left[\overline{a}_{i,1}^{k+1}, \overline{a}_{i,2}^{k+1}, \cdots, \overline{a}_{i,2d_k}^{k+1}\right], \\ \overline{a}_{i,j}^{k+1} &= a_{i,j}^{k+1} \text{ and } \overline{a}_{i,j+d_k}^{k+1} = s_j^k \times a_{i,j}^{k+1}, j \in \{1, \cdots, d_k\}.\end{aligned} \tag{10}$$

3 Overview of our attack

3.1 Adversarial Goals and assumptions

There are two parties in a model parameter recovery attack: the adversary and the Oracle. The adversary generates an input X, queries the Oracle, and obtains the feedback. By collecting a sufficient number of input-feedback pairs, the adversary recovers a set of parameters $\widehat{\theta}$, with the goal that $f_{\widehat{\theta}}$ and f_θ are functionally equivalent. The $\widehat{\ }$ symbol is used to denote a finally recovered parameter.

Assumptions. We make the following assumptions about the Oracle and the capabilities of the adversary. (1) **Architecture knowledge:** we require knowledge of the architecture of the neural network. (2) **Chosen input:** we can feed arbitrary inputs from $\mathcal{X} = \mathbb{R}^{d_0}$. (3) **Precise computations:** f_θ is specified and evaluated using 64-bit floating point arithmetic. (4) **PReLU activations:** all activation functions $\sigma_{k,j}$ are the PReLU functions. (5) **Hard-label setting:** the feedback is the hard label.

Compared with existing attacks [8,20] on PReLU neural networks, we remove the slope and structure restrictions. Specifically, this paper focuses on expansive PReLU neural networks and imposes no restrictions on the neuron slopes. For the convenience of introducing our work, we take a typical type of n-deep expansive networks as an example, the dimensions $d_0, d_1, \cdots, d_{n+1}$ of which satisfy

$$d_{k-1} < d_k \text{ for } k \in \{1, \cdots, n-1\}. \tag{11}$$

3.2 Attack Framework

New Framework. At a high level, our attack framework for an n-deep expansive PReLU FCN is as follows. The model parameters are still recovered layer by layer. In each layer, for the parameter recovery of each neuron, only the step of neuron signature recovery is kept. The step of neuron sign recovery is removed by exploiting the new network isomorphism named flip-and-scaling (see Section 4).

The joint neuron signature recovery method (see Section 6) is applied to layer $k+1$ for $k \in \{1, 2, \cdots, n-1\}$. Note that the neuron signature recovery method in [5] does not apply to layer $k+1$ due to the linear constraints (see Section 5). Before running the joint neuron signature recovery, the neuron splitting technique is first applied to layer k. In layer $k+1$, the extended neuron signatures $\overline{A}_i^{k+1}$ are recovered by the joint neuron signature recovery, which further leads to the recovery of $\tilde{A}_i^{k+1}$, $\tilde{b}_i^{k+1}$, and $\tilde{s}_j^k$ (the d_k neuron slopes in layer k).

In layer 1, the neuron signature recovery method in [5] is applied to recover the neuron signatures $\tilde{A}_i^1$ and biases $\tilde{b}_i^1$. As for the last linear layer, it is directly handled by the neuron splitting technique and the methods in [4], refer to [4,20]. For simplicity, the remainder of this paper will omit the description of the first and last layers.

Core Process in Recovering Layer $k+1$. For the typical type of expansive PReLU neural networks (see Eq. (11)), the core process in recovering layer $k+1$ where $k \in \{1, 2, \cdots, n-1\}$ is as follows:

1. Apply the neuron splitting technique to layer k.
2. Recover the following d_{k+1} extended neuron signatures in layer $k+1$ by the joint neuron signature recovery

$$\left[\tilde{a}_{i,1}^{k+1}, \tilde{a}_{i,2}^{k+1}, \cdots, \tilde{a}_{i,2d_k}^{k+1}\right], \ i \in \{1, \cdots, d_{k+1}\}, \tag{12}$$

where

$$\left(\tilde{a}_{i,j}^{k+1}, \tilde{a}_{i,j+d_k}^{k+1}\right) = \begin{cases} \left(\widehat{a}_{i,j}^{k+1}, s_j^k \times \widehat{a}_{i,j}^{k+1}\right), & \text{if } \tilde{A}_j^k = \widehat{A}_j^k, \\ \left(-s_j^k \times \widehat{a}_{i,j}^{k+1}, -\widehat{a}_{i,j}^{k+1}\right), & \text{if } \tilde{A}_j^k = -\widehat{A}_j^k. \end{cases} \tag{13}$$

3. Set the neuron signatures $\tilde{A}_i^{k+1}$ in layer $k+1$ and slopes $\tilde{s}_j^k$ in layer k to

$$\begin{aligned} \tilde{A}_i^{k+1} &= \left[\tilde{a}_{i,1}^{k+1}, \tilde{a}_{i,2}^{k+1}, \cdots, \tilde{a}_{i,d_k}^{k+1}\right], \text{ for } i \in \{1, \cdots, d_{k+1}\} \\ \tilde{s}_j^k &= \frac{\tilde{a}_{i,j+d_k}^{k+1}}{\tilde{a}_{i,j}^{k+1}}, \text{ for } j \in \{1, \cdots, d_k\}. \end{aligned} \tag{14}$$

Note that we have $\tilde{s}_j^k = s_j^k$ if $\tilde{A}_j^k = \widehat{A}_j^k$, or $\tilde{s}_j^k = \frac{1}{s_j^k}$ if $\tilde{A}_j^k = -\widehat{A}_j^k$. Return $\tilde{A}_i^{k+1}$ and $\tilde{s}_j^k$ as the finally recovered parameters .

In Section 6, we propose two improvements to enhance the joint neuron signature recovery. Therefore, the core process described here corresponds to Stage 2 of Algorithm 1.

4 Break Slope Restriction by New Network Isomorphism

As introduced in Section 1, the slope restriction in existing attacks [8, 20] is attributed to the neuron sign recovery. In this section, we first introduce the network isomorphisms of ReLU neural networks, and analyze why these network isomorphisms make the step of neuron sign recovery necessary for ReLU neural networks. Then, we propose a new network isomorphism for PReLU neural networks. At last, based on the new network isomorphism, we show how to break the slope restriction ($s_j^k < 1$) by removing the step of neuron sign recovery for PReLU neural networks.

4.1 Why sign recovery is necessary in ReLU neural networks

Network Isomorphisms in ReLU Neural Networks. ReLU neuron networks have two types of isomorphism: **Permutation** and **Scaling**. Permutation only swaps the order of neurons in a layer, and does not change any model parameters. Therefore, we do not introduce more about it.

Scaling. Consider a neuron η_j^k with incoming weights A_j^k, bias b_j^k, and outgoing weights $A_{:,j}^{k+1}$. Multiplying the incoming weights and bias by a positive factor $c > 0$ while multiplying the outgoing weights by a factor $\frac{1}{c}$, is referred to as scaling isomorphism:

$$A_j^k \leftarrow c \cdot A_j^k;\ b_j^k \leftarrow c \times b_j^k;\ A_{:,j}^{k+1} \leftarrow \frac{1}{c} \cdot A_{:,j}^{k+1}. \tag{15}$$

Fig. 2 shows the system diagram of the scaling isomorphism.

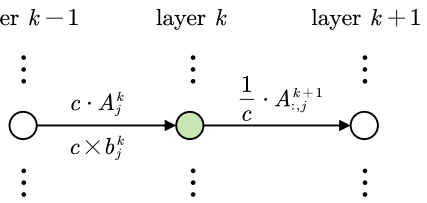

Fig. 2. The system diagram of the scaling isomorphism. For the j-th neuron in layer k, denoted by the solid circle, if the incoming weights A_j^k and bias b_j^k are multiplied by a positive factor $c > 0$, and the outgoing weights $A_{:,j}^{k+1}$ are divided by $\frac{1}{c}$, the raw output of the network is not affected.

The Influence of Scaling Isomorphism on The Raw Output. In [22], Rolnick et al. proved that scaling isomorphism does not affect the raw output of ReLU neural networks. To help understand the limitation of the scaling isomorphism introduced later, we first recall why the raw output is not affected by the scaling isomorphism.

Consider the j-th neuron η_j^k in layer k again. Suppose that the scaling transformation in Eq. (15) is applied to η_j^k, and we focus on whether the preactivation input y_p^{k+1} of neurons η_p^{k+1} in the next layer will be changed or not.

Before applying the scaling transformation to η_j^k, we have

$$y_p^{k+1} = \sum_{i=1}^{d_k} y_{(p,i)}^{k+1} + b_p^{k+1}, \tag{16}$$

where the contribution of η_j^k is denoted by $y_{(p,j)}^{k+1}$. The value of $y_{(p,j)}^{k+1}$ is

$$y_{(p,j)}^{k+1} = \begin{cases} a_{p,j}^{k+1}(A_j^k Z^{k-1} + b_j^k), & \text{if } A_j^k Z^{k-1} + b_j^k \geq 0, \\ 0, & \text{if } A_j^k Z^{k-1} + b_j^k < 0, \end{cases} \tag{17}$$

where $a_{p,j}^{k+1}$ is the connection weight between η_j^k and η_p^{k+1} (the p-th element of $A_{:,j}^{k+1}$), and Z^{k-1} is the output of layer $k-1$ (refer to Section 2).

After applying the scaling transformation to η_j^k, the contribution of η_j^k becomes $\left(y_{(p,j)}^{k+1}\right)_{\text{new}}$ where

$$\left(y_{(p,j)}^{k+1}\right)_{\text{new}} = \begin{cases} \left(\frac{1}{c} \times a_{p,j}^{k+1}\right)\left(c \cdot A_j^k Z^{k-1} + c \times b_j^k\right), & \text{if } A_j^k Z^{k-1} + b_j^k \geq 0, \\ 0, & \text{if } A_j^k Z^{k-1} + b_j^k < 0. \end{cases} \tag{18}$$

Note that $\left(\frac{1}{c} \times a_{p,j}^{k+1}\right)\left(c \cdot A_j^k Z^{k-1} + c \times b_j^k\right) = a_{p,j}^{k+1}(A_j^k Z^{k-1} + b_j^k)$. Therefore, we have $\left(y_{(p,j)}^{k+1}\right)_{\text{new}} = y_{(p,j)}^{k+1}$, which means that the preactivation input y_p^{k+1} of η_p^{k+1} is not affected by the application of the scaling isomorphism to η_j^k. Furthermore, the final raw output of the ReLU neural network is not affected.

The Limitation of Scaling Isomorphism. The limitation of scaling isomorphism lies in its requirement that scaling factors be strictly positive. Consequently, neuron sign recovery becomes a necessary step for cryptanalytic extraction attacks on ReLU neural networks. To understand this causal relationship, let us see what happens if the scaling factors are negative.

Consider the scaling transformation as shown in Eq. (15), and suppose that the scaling factors are $-c$ and $-\frac{1}{c}$, instead of c and $\frac{1}{c}$ where $c > 0$. After applying such new scaling transformations to η_j^k, the contribution of η_j^k to the preactivation input y_p^{k+1} becomes

$$\left(y_{(p,j)}^{k+1}\right)_{\text{new}} = \begin{cases} 0, & \text{if } A_j^k Z^{k-1} + b_j^k \geq 0, \\ \left(-\frac{1}{c} \times a_{p,j}^{k+1}\right)\left(-c \cdot A_j^k Z^{k-1} - c \times b_j^k\right), & \text{if } A_j^k Z^{k-1} + b_j^k < 0. \end{cases} \tag{19}$$

Compare Eq. (17) and Eq. (19), we know that $\left(y_{(p,j)}^{k+1}\right)_{\text{new}} \neq y_{(p,j)}^{k+1}$ always holds (except for the case $A_j^k Z^{k-1} + b_j^k = 0$), which changes the preactivation input y_p^{k+1}, and further makes the raw output of the network be changed with a non-negligible or even high probability.

Now, recall that the neuron signature recovery in existing attacks only recovers a weight vector $\tilde{A}_j^k \in \{\widehat{A}_j^k, -\widehat{A}_j^k\}$, while our target is to obtain $\widehat{A}_j^k$. If $\tilde{A}_j^k = \widehat{A}_j^k$ (respectively, $\tilde{A}_j^k = -\widehat{A}_j^k$), it is equivalent to applying the scaling transformation with positive (respectively, negative) scaling factors to the neuron η_j^k. To ensure the same contribution of η_j^k to the preactivation input of neurons in the next layer, one must determine whether $\tilde{A}_j^k = \widehat{A}_j^k$ or not. Thus, neuron sign recovery is a necessary step for ReLU neural networks.

4.2 New Network Isomorphism in PReLU Neural Networks

Permutation and Scaling in PReLU Neural Networks. At first, the two types of network isomorphisms, permutation and scaling introduced in Section 4.1, are still applicable to PReLU neural networks [8].

There is an important tip about the scaling isomorphism in PReLU neural networks. Consider the j-th neuron η_j^k in layer k. When applying the scaling transformation to it, the neuron slope s_j^k is not changed, and only its incoming weights, bias, and outgoing weights need to be scaled [8].

Flip-and-Scaling: New Network Isomorphism. Except for permutation and scaling, we find a new network isomorphism for PReLU neural networks, which is named flip-and-scaling. This new network isomorphism makes the neuron sign recovery unnecessary for PReLU neural networks.

As shown in the left subfigure of Fig. 3, the flip-and-scaling isomorphism makes the following transformation to the parameters related to the neuron η_j^k:

$$A_j^k \leftarrow -A_j^k;\ b_j^k \leftarrow -b_j^k;\ A_{:,j}^{k+1} \leftarrow -s_j^k \cdot A_{:,j}^{k+1};\ s_j^k \leftarrow \frac{1}{s_j^k}. \tag{20}$$

Concretely, we first flip the overall sign of incoming weight A_j^k, bias b_j^k, and outgoing weight $A_{:,j}^{k+1}$, and then multiply the outgoing weight by s_j^k, where s_j^k is the neuron slope. At last, we replace the slope s_j^k of the PReLU activation by its reciprocal $\frac{1}{s_j^k}$.

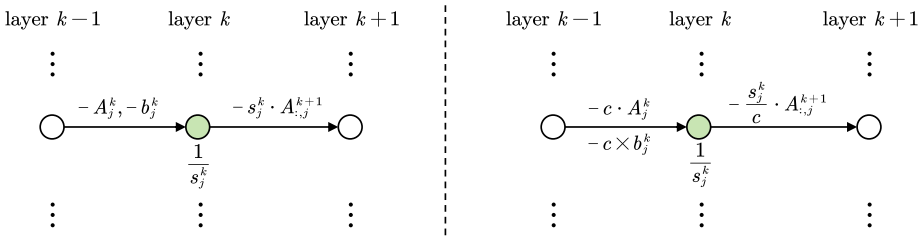

Fig. 3. The system diagram of the flip-and-scaling isomorphism (left figure), and its combination with the scaling isomorphism (right figure). For the j-th neuron in layer k, denoted by the solid circle, if the incoming weights and bias are multiplied by -1 (or $-c$ where $c > 0$), the initial neuron slope is replaced by its reciprocal $\frac{1}{s_j^k}$, and the outgoing weights are multiplied by $-s_j^k$ (or $-\frac{s_j^k}{c}$), the raw output of the PReLU neural network is not affected.

The Influence of Flip-and-Scaling on the Raw Output. Like the analysis of the scaling isomorphism in ReLU neural networks, we focus on the contribution $y_{(p,j)}^{k+1}$ of η_j^k to the preactivation input y_p^{k+1} of the neuron η_p^{k+1}.

Before applying the flip-and-scaling transformation to η_j^k, we have

$$y_{(p,j)}^{k+1} = \begin{cases} a_{p,j}^{k+1} \times (A_j^k Z^{k-1} + b_j^k), & \text{if } A_j^k Z^{k-1} + b_j^k \geq 0, \\ a_{p,j}^{k+1} \times (s_j^k \times (A_j^k Z^{k-1} + b_j^k)), & \text{if } A_j^k Z^{k-1} + b_j^k < 0. \end{cases} \tag{21}$$

After applying the flip-and-scaling transformation, the contribution becomes

$$\left(y_{(p,j)}^{k+1}\right)_{\text{new}} = \begin{cases} -s_j^k \times a_{p,j}^{k+1} \times (\frac{1}{s_j^k} \times (-A_j^k Z^{k-1} - b_j^k)), & \text{if } A_j^k Z^{k-1} + b_j^k \geq 0, \\ -s_j^k \times a_{p,j}^{k+1} \times (-A_j^k Z^{k-1} - b_j^k), & \text{if } A_j^k Z^{k-1} + b_j^k < 0. \end{cases} \tag{22}$$

Interestingly, we still have $\left(y_{(p,j)}^{k+1}\right)_{\text{new}} = y_{(p,j)}^{k+1}$. Thus, the flip-and-scaling transformation, applied to the neuron η_j^k in layer k, does not change the preactivation input y_p^{k+1} of neurons in the next layer. Furthermore, the raw output of PReLU neural networks is not affected.

Combining Scaling with Flip-and-Scaling. The flip-and-scaling isomorphism can be combined with the scaling isomorphism. Concretely, as shown in the right subfigure of Fig. 3, we make the following transformation to the neuron η_j^k:

$$A_j^k \leftarrow -c \cdot A_j^k;\ b_j^k \leftarrow -c \times b_j^k;\ A_{:,j}^{k+1} \leftarrow \frac{-s_j^k}{c} \cdot A_{:,j}^{k+1};\ s_j^k \leftarrow \frac{1}{s_j^k}, \tag{23}$$

where $c > 0$. This combination does not change the preactivation input y_p^{k+1} of neurons in the next layer, nor the raw output of PReLU neural networks.

Experimental Verification. We have experimentally verified the new network isomorphism as follows. Starting from a PReLU network f_θ, for each neuron η_i^k in every layer (from layer 1 to the last), we randomly selected one of the isomorphisms (scaling, flip-and-scaling, or a combination) and applied it to that neuron, obtaining a new network $f_{\widehat{\theta}}$. Then, feed 10^5 random network inputs X to both networks and compare their raw outputs. For all tested networks (e.g., 20-15-10-5), the outputs were identical up to numerical precision, confirming that f_θ and $f_{\widehat{\theta}}$ are functionally equivalent.

4.3 Break Slope Restriction by Removing Sign Recovery

Impact of Flip-and-Scaling on Neuron Sign Recovery. Now, let us reconsider whether the step of neuron sign recovery is necessary for PReLU neural networks.

Again, recall that the neuron signature recovery of existing attacks recovers a weight vector $\tilde{A}_j^k \in \{\widehat{A}_j^k, -\widehat{A}_j^k\}$. Due to the two network isomorphisms, scaling

and flip-and-scaling, we do not need to recover the neuron sign anymore, if the following conditions on the neuron slope are satisfied:

$$\widehat{s}_j^k = \begin{cases} s_j^k, & \text{if } \tilde{A}_j^k = \widehat{A}_j^k, \\ \frac{1}{s_j^k}, & \text{if } \tilde{A}_j^k = -\widehat{A}_j^k. \end{cases} \tag{24}$$

Besides, when $\tilde{A}_j^k = -\widehat{A}_j^k$, the scaling factor $-s_j^k$ should be included in $\widehat{a}_{p,j}^{k+1}$, such as $\widehat{a}_{p,j}^{k+1} = -s_j^k \times a_{p,j}^{k+1}$.

Overall, when $\tilde{A}_j^k = \widehat{A}_j^k$ (respectively, $\tilde{A}_j^k = -\widehat{A}_j^k$), the finally recovered slope should be $\widehat{s}_j^k = s_j^k$ (respectively, $\widehat{s}_j^k = \frac{1}{s_j^k}$), corresponding to the scaling isomorphism (respectively, the flip-and-scaling isomorphism, or its combination with scaling).

Remove Sign Recovery and Compute Slope Adaptively. The above conditions, as presented in Eq. (24), are satisfied by utilizing the neuron splitting technique [8] introduced in Section 2.3.

For ease of understanding, let us first take the j-th neuron in layer k as an example. After $\tilde{A}_j^k$ is recovered, we apply the neuron splitting technique, and perform the neuron signature recovery for η_p^{k+1}, under the assumption that $\tilde{A}_j^k = \widehat{A}_j^k$, which will return a pair of weights $(\tilde{a}_{p,j}^{k+1}, \tilde{a}_{p,j+d_k}^{k+1})$. Regarding whether the assumption that $\tilde{A}_j^k = \widehat{A}_j^k$ holds or not, one has

$$(\widetilde{a}_{p,j}^{k+1}, \widetilde{a}_{p,j+d_k}^{k+1}) = \begin{cases} (\widehat{a}_{p,j}^{k+1}, s_j^k \times \widehat{a}_{p,j}^{k+1}) & \text{if } \tilde{A}_j^k = \widehat{A}_j^k, \\ (-s_j^k \times \widehat{a}_{p,j}^{k+1}, -\widehat{a}_{p,j}^{k+1}) & \text{if } \tilde{A}_j^k = -\widehat{A}_j^k. \end{cases} \tag{25}$$

Next, we compute the value of the slope $\widehat{s}_j^k$ as $\widehat{s}_j^k = \frac{\widetilde{a}_{p,j+d_k}^{k+1}}{\widetilde{a}_{p,j}^{k+1}}$, which is consistent with the expected value summarized in Eq. (24). As a result, we do not need to recover the neuron sign. Besides, the recovery of a neuron slope is not restricted by its value. Thus, there is no need to make any restrictions to the neuron slopes.

Now, consider the d_k neurons in layer k simultaneously. After recovering all the d_k neuron signatures in layer k, we directly go to the neuron signature recovery of layer $k+1$, and the target is to recover the following extended weight vectors

$$\left[\widetilde{a}_{p,1}^{k+1}, \widetilde{a}_{p,2}^{k+1}, \cdots, \widetilde{a}_{p,2d_k}^{k+1}\right] \tag{26}$$

where the values of each pair $(\widetilde{a}_{p,j}^{k+1}, \widetilde{a}_{p,j+d_k}^{k+1})$ are presented in Eq. (25). Based on this extended weight vector, one obtains all the d_k neuron slopes $\widehat{s}_j^k$ for $1 \le j \le d_k$, where $\widehat{s}_j^k = \frac{\widetilde{a}_{p,j+d_k}^{k+1}}{\widetilde{a}_{p,j}^{k+1}}$.

5 Signature Recovery Challenge Caused by Additional Linear Constraints

To recover the extended weight vectors in layer $k+1$ as shown in Eq.(26), a natural idea is to apply the hard-label based signature recovery method [5] introduced in Section 2.2, originally designed for ReLU neural networks. However, we find that it does not apply to expansive PReLU neural networks after we apply the neuron splitting technique to layer k, which is attributed to the additional linear constraints on the output of the extended layer k. This section explains how these linear constraints are generated and presents their exact number.

The Vital Task of Applying the Neuron Signature Recovery Method in [5]. Before presenting our analysis about linear constraints, we first recall the vital task of applying the neuron signature recovery method in [5] to PReLU neural networks.

Consider the signature recovery of the i-th neuron η_i^{k+1} in layer $k+1$. As shown in Fig. 1, after splitting each neuron of layer k into a pair of neurons, the goal is to recover the extended neuron signature

$$\overline{A}_i^{k+1} = \left[a_{i,1}^{k+1}, a_{i,2}^{k+1}, \cdots, a_{i,d_1}^{k+1}, s_1^k \times a_{i,1}^{k+1}, s_2^k \times a_{i,2}^{k+1}, \cdots, s_{d_k}^k \times a_{i,d_k}^{k+1},\right]. \quad (27)$$

If we try to recover $\overline{A}_i^{k+1}$ by the hard-label based signature recovery method in [5], the vital task is to recover the following $(2d_k-1)$-dimensional affine space (by collecting dual points corresponding to the neuron η_i^{k+1})

$$V = \left\{ \overline{Z}^k \,\middle|\, \overline{A}_i^{k+1} \overline{Z}^k + b_i^{k+1} = 0 \right\}, \quad (28)$$

where $\overline{Z}^k = \left[\overline{z}_1^k, \overline{z}_2^k, \cdots, \overline{z}_{2d_k}^k\right]^\top$.

However, when there are additional linear constraints on $\overline{Z}^k$, the dimension of the affine space reconstructed in practice would be smaller than $2d_k-1$, making the neuron signature recovery method in [5] fail.

5.1 Warm-Up: Linear Constraints on the First Layer

We start the analysis from layer 1. Suppose that the neuron splitting technique is applied to layer 1, and we care about the number of linear constraints on $\overline{Z}^1$, the output of the extended layer 1.

Consider that $\overline{Z}^1$ is determined by Y^1, the preactivation input of layer 1, and $Y^1 = A^1 X + B^1$ where X is the input of the network, we first figure out the number of linear constraints on Y^1.

The Number of Linear Constraints on Y^1. Since $A^1 \in \mathbb{R}^{d_1 \times d_0}$, the dimension of the left null space of A^1 is $d_1 - \text{rank}(A^1)$. Denote by $m_1 = d_1 - \text{rank}(A^1)$.

Let $C_1^1, C_2^1, \ldots, C_{m_1}^1 \in \mathbb{R}^{d_1}$ be a set of basis vectors of the left null space of A^1, i.e., $\left(C_p^1\right)^\top A^1 = \mathbf{0}$ for $p = 1, \ldots, m_1$. Due to the equation $Y^1 = A^1 X + B^1$, we have

$$\left(C_p^1\right)^\top (Y^1 - B^1) = \left(C_p^1\right)^\top Y^1 - \left(C_p^1\right)^\top B^1 = \mathbf{0}. \tag{29}$$

Thus, the preactivation input Y^1 of layer 1 is subject to m_1 linear constraints.

For simplicity, we rewrite the m_1 linear constraints as $C^1 Y^1 = CB^1$ where

$$C^1 = [(C_1^1)^\top; (C_2^1)^\top; \cdots; (C_{m_1}^1)^\top] \in \mathbb{R}^{m_1 \times d_1}. \tag{30}$$

The Number of Linear Constraints On $\overline{Z}^1$. According to the introduction of the neuron splitting technique [8] in Section 2.3, we have

$$\overline{z}_j^1 = \max(y_j^1, 0) \text{ and } \overline{z}_{j+d_1}^1 = \min(y_j^1, 0), \tag{31}$$

which leads to the relations as follows

$$\overline{z}_j^1 + \overline{z}_{j+d_1}^1 = y_j^1, \ j \in \{1, \cdots, d_1\}. \tag{32}$$

Expressing the above relations in a matrix form, we have

$$Y^1 = H^1 \overline{Z}^1 = \begin{bmatrix} 1\,0\cdots 0\,1\,0\cdots 0 \\ 0\,1\cdots 0\,0\,1\cdots 0 \\ \vdots\,\vdots\,\ddots\,\vdots\,\vdots\,\vdots\,\ddots\,\vdots \\ 0\,0\cdots 1\,0\,0\cdots 1 \end{bmatrix} \overline{Z}^1 = [I_{d_1}, I_{d_1}]\, \overline{Z}^1, \tag{33}$$

where $H^1 \in \mathbb{R}^{d_1 \times 2d_1}$, and I_{d_1} is an $d_1 \times d_1$ identity matrix. Note that $H^1 = [I_{d_1}, I_{d_1}]$.

Since $Y^1 = H^1 \overline{Z}^1$ and $C^1 Y^1 = C^1 B^1$, we further get

$$C^1 H^1 \overline{Z}^1 = C^1 B^1. \tag{34}$$

Note that $C^1 H^1 = \left[C^1, C^1\right]$. Therefore, the rank of $C^1 H^1$ is $\text{rank}(C^1 H^1) = \text{rank}(C^1) = m_1$, meaning that $\overline{Z}^1$ is also subject to $m_1 = d_1 - \text{rank}(A^1)$ linear constraints.

5.2 Linear Constraints on Deeper Layers

Now, we study the number of linear constraints on $\overline{Z}^k$, the output of the extended layer k (after applying the neuron wiggle technique to layer k). Note that $\overline{Z}^k$ is determined by Y^k and $Y^k = A^k Z^{k-1} + B^k$, where $A^k \in \mathbb{R}^{d_k \times d_{k-1}}$.

Before analyzing the number of linear constraints on Y^k and $\overline{Z}^k$, we need to figure out whether there are linear constraints on Z^{k-1}. To answer this question, we propose the follow proposition.

Proposition 1. *Consider layer i in a PReLU neural network. Under the assumption that for any $j \in \{1, 2, \cdots, d_i\}$, $y_j^i > 0$ does not always holds and $y_j^i < 0$ does not always hold either, there are no linear constraints (denoted by $Q^\top Z^i = c$) on the output Z^i of layer i, where $Q \in \mathbb{R}^{d_i}$ and $Q \neq \mathbf{0}$, and c is a constant.*

Proof. The formal proof and an empirical verification are given in Appendix A.

Based on Proposition 1, there are no linear constraints on Z^{k-1}, which is further verified by the support experiments in Section 5.3. Then, the number of linear constraints on Y^k is only determined by the left null space of A^k.

The Number of Linear Constraints on Y^k. Since $A^k \in \mathbb{R}^{d_k \times d_{k-1}}$, the dimension of the left null space of A^k is $d_k - \operatorname{rank}(A^k)$.

Denote by $m_k = d_k - \operatorname{rank}(A^k)$. Let $C_1^k, C_2^k, \ldots, C_{m_k}^k \in \mathbb{R}^{d_k}$ be a set of basis vectors of the left null space of A^k, i.e., $\left(C_p^k\right)^\top A^k = \mathbf{0}$ for $p \in \{1, \cdots, m_k\}$. Due to the equation $Y^k = A^k Z^{k-1} + B^k$, we have

$$\left(C_p^k\right)^\top (Y^k - B^k) = \left(C_p^k\right)^\top Y^k - \left(C_p^k\right)^\top B^k = \mathbf{0}, \text{for } p \in \{1, \cdots, m_k\}, \tag{35}$$

Thus, the preactivation input Y^k of layer k is subject to m_k linear constraints.

Let

$$C^k = [(C_1^k)^\top; (C_2^k)^\top; \cdots; (C_{m_k}^k)^\top] \in \mathbb{R}^{m_k \times d_k}, \tag{36}$$

and we rewrite the m_k linear constraints as $C^k Y^k = C^k B^k$.

The Number of Linear Constraints On $\overline{Z}^k$. Recall that

$$\begin{aligned} \overline{Z}^k &= \left[\bar{z}_1^k, \bar{z}_2^k, \cdots, \bar{z}_{2d_k}^k\right]^\top, \\ \bar{z}_j^k &= \max(y_j^k, 0) \text{ and } \bar{z}_{j+d_k}^k = \min(y_j^k, 0), j \in \{1, \cdots, d_k\}. \end{aligned} \tag{37}$$

The above relations are equivalent to $Y^k = H^k \overline{Z}^k$ where $H^k \in \mathbb{R}^{d_k \times 2d_k}$ and

$$H^k = \begin{bmatrix} 1\,0\cdots 0\,1\,0\cdots 0 \\ 0\,1\cdots 0\,0\,1\cdots 0 \\ \vdots\,\vdots\,\ddots\,\vdots\,\vdots\,\vdots\,\ddots\,\vdots \\ 0\,0\cdots 1\,0\,0\cdots 1 \end{bmatrix} = [I_{d_k}, I_{d_k}]\,. \tag{38}$$

Here, the notation I_{d_k} stands for an $d_k \times d_k$ identity matrix.

Since $Y^k = H^k \overline{Z}^k$ and $C^k Y^k = C^k B^k$, we further get

$$C^k H^k \overline{Z}^k = C^k B^k. \tag{39}$$

Note that $C^k H^k = \left[C^k, C^k\right]$. Therefore, the rank of $C^k H^k$ is $\operatorname{rank}(C^k H^k) = \operatorname{rank}(C^k) = m_k$, meaning that $\overline{Z}^k$ is subject to $m_k = d_k - \operatorname{rank}(A^k)$ linear constraints.

Combined with the basic case in Section 5.1, we know that the number of linear constraints on the extended output of each layer is $m_k = d_k - \operatorname{rank}(A^k)$.

5.3 Impact on The Affine Space Reconstructed in Practice

The Influence of The Relation Between d_k And d_{k-1}. As introduced, there are $m_k = d_k - \text{rank}(A^k)$ linear constraints on $\overline{Z}^k$, the output of the extended layer k.

When $d_k > d_{k-1}$, we have $\text{rank}(A^k) < d_k$ (equally, $m_k > 0$) since $A^k \in \mathbb{R}^{d_k \times d_{k-1}}$. When $d_k \leq d_{k-1}$, we get $\text{rank}(A^k) \leq d_k$. If the equation $\text{rank}(A^k) = d_k$ further holds, then $m_k = 0$.

The Dimension of The The Affine Space Reconstructed in Practice. Denoted by V_{real} the affine space reconstructed in practice. Since there are m_k linear constraints on $\overline{Z}^k$, the dimension of V_{real} is expected to be $2d_k - 1 - m_k = d_k + \text{rank}(A^k) - 1$ in general.

A Special Case in Theory. Theoretically, regarding the dimension of the reconstructed space V_{real}, there is an exception as follows. When the constraint $\overline{A}_i^{k+1}\overline{Z}^k + b_i^{k+1} = 0$ can be derived from the m_k linear constraints $C^k H^k \overline{Z}^k = C^k B^k$. The real dimension of V_{real} would be $d_k + \text{rank}(A^k)$, instead of $d_k + \text{rank}(A^k) - 1$. But the probability that this special case occurs approaches 0, since it is equivalent to randomly selecting an item from a $2d_k$-dimensional space, and this item is also in an m_k-dimensional subspace. Therefore, we omit this special case in the following paper.

Experimental Verification. Practical experiments are performed on three PReLU neural networks with randomly generated model parameters, to verify the conclusion that the number of constraints on $\overline{Z}^k$ is $m_k = d_k - \text{rank}(A^k)$. The affine space reconstructing method in [5] is applied to reconstruct the space V_{real}. Table 1 summarizes the experimental results, which fully support our conclusion.

Table 1. The dimension of the space V_{real} reconstructed in practice over $\overline{Z}^k$. E.D.: Expected Dimension. R.D.: Real Dimension

structure	$k=1$					$k=2$				
	d_k	$\text{rank}(A^k)$	E.D.	R.D.	consistent?	d_k	$\text{rank}(A^k)$	E.D.	R.D.	consistent?
45-50-55-60-10	50	45	94	94	Yes	55	50	104	104	Yes
30-20-25-30-10	20	20	39	39	Yes	25	20	44	44	Yes
20-25-20-15-10	25	20	44	44	Yes	20	20	39	39	Yes

Besides, the special case in theory does not occur. In our experiments (in Section 7) based on 800 expansive PReLU neural networks, we do not observe the special case either.

6 Joint Neuron Signature Recovery

This section introduces a joint neuron signature recovery method to recover layer $k+1$ (specifically, recover neuron signatures in layer $k+1$ and slopes in layer k) in the hard-label setting, which applies to the case of $d_k > d_{k-1}$.

6.1 Key Observations

Based on the analysis in Section 5.2, for each neuron η_i^{k+1} in layer $k+1$, we can reconstruct a $(d_k + \text{rank}(A^k) - 1)$-dimensional affine space (denoted by V_i) over $\overline{Z}^k$, by collecting a sufficient number of dual points corresponding to η_i^{k+1}.

Moreover, the affine space V_i reconstructed in practice is as follows

$$V_i = \left\{ \overline{Z}^k \,\middle|\, \overline{A}_i^{k+1} \overline{Z}^k + b_i^{k+1} = 0, C^k H^k \overline{Z}^k = C^k B^k \right\}, i \in \{1, 2, \cdots, d_{k+1}\}, \quad (40)$$

where $C^k H^k \overline{Z}^k = C^k B^k$ is the m_k linear constraints (see Eq. (39)).

Relation Between $\overline{A}_i^{k+1}$ and The Orthogonal Complement of V_i. According to Eq, (40), the dimension of $V_i^\perp$, the orthogonal complement space of V_i, is $m_k + 1$. And the extended weight vector $\overline{A}_i^{k+1} \in \mathbb{R}^{2d_k}$ to be recovered is in the (m_k+1)-dimensional space $V_i^\perp$.

Common Subspace. According to Eq, (40), we know that all orthogonal complement spaces $V_i^\perp$ (for $i \in \{1, \cdots, d_{k+1}\}$) contain the same m_k-dimensional subspace (denoted by W), such that $V_i^\perp = W \oplus \text{span}\{U_0^i\}$, where $C^k H^k$ spans W. Denote by $\{U_1, \cdots, U_{m_k}\}$ a set of basis vectors of W. Then, an explicit construction of $U_1, \cdots, U_{m_k}$ from $C^k H^k$ is given in Appendix C.

6.2 A Heuristic Joint Recovery Method

Joint Recovery Framework. Choose a subset of N neurons in layer $k+1$, where $N \leqslant d_{k+1}$. For simplicity, suppose that the chosen neurons are η_1^{k+1}, η_2^{k+1}, $\cdots$, η_N^{k+1}.

For each chosen neuron η_i^{k+1}, we reconstruct the affine space V_i and randomly generate a set of basis vectors [7] $\{U_0^i, U_1^i, \cdots, U_{m_k}^i\}$ of its orthogonal complement space $V_i^\perp$. Then the extended weight vector $\overline{A}_i^{k+1}$ can be parameterized as

$$(\overline{A}_i^{k+1})^\top = t_0^i \cdot U_0^i + t_1^i \cdot U_1^i + \cdots + t_{m_k}^i \cdot U_{m_k}^i, \text{ for } 1 \leq i \leq N \quad (41)$$

[7] Although a common subspace exists and $\{U_1^i, U_2^i, \cdots, U_{m_k}^i\}, i \in \{1, \cdots, N\}$ can be the common vectors $U_1, U_2, \cdots, U_{m_k}$ constructed from A^k, randomly generated bases span their respective $V_i^\perp$ and are therefore equally valid. This simpler implementation does not affect correctness.

where $t_l^i \in \mathbb{R}$ for $l \in \{0, \cdots, m_k\}$ are the unknown coefficients. By the scaling and flip-and-scaling isomorphisms, we can fix $t_0^i = 1$ for each i, leaving m_k unknowns per neuron.

The d_k unknown slopes $\tilde{s}_j^k$ must satisfy:

$$\tilde{s}_j^k = \frac{\overline{a}_{i,j+d_k}^{k+1}}{\overline{a}_{i,j}^{k+1}}, \text{ for } 1 \leq j \leq d_k, \ 1 \leq i \leq N. \tag{42}$$

Substituting the parameterization (Eq.(41)) yields a nonlinear system with $N \cdot m_k + d_k$ unknowns and $N \cdot d_k$ equations.

After building such a system, we solve it by a least square solver (all the unknowns are initialized to random values), which aims to minimize the following loss function:

$$\text{loss} = \sum_{j=1}^{d_k} \sum_{i=1}^{N} \left(\tilde{s}_j^k \times \overline{a}_{i,j}^{k+1} - \overline{a}_{i,j+d_k}^{k+1} \right)^2. \tag{43}$$

The true model parameters to be recovered, including the N extended weight vectors $\overline{A}_i^{k+1}$ and the d_k slopes $\widehat{s}_j^k$, will lead to a loss of 0, the minimal loss.

Necessary Condition for Solving the Non-linear System. To make the system solvable, we need $N \cdot d_k \geqslant N \cdot m_k + d_k$, i.e., $N \cdot \text{rank}(A^k) \geqslant d_k$. Since $N \leqslant d_{k+1}$, a necessary condition is $d_{k+1} \cdot \text{rank}(A^k) \geqslant d_k$, which is easily satisfied for expansive networks.

We emphasize that the necessary condition above does not guarantee a unique solution. The nonlinear system may have multiple solutions. Hence, strictly speaking, the joint recovery method is heuristic in nature. In the next subsection we introduce two improvements (residual analysis and progressive parameter decoupling) that make the method robust and obtain better-quality recovery in practice.

Limitation. As noted, the joint recovery method is heuristic in nature. Nevertheless, it works well in practice, especially after adopting the two improvements, as verified across several hundred networks in Section 7. Proving the uniqueness of the solution or developing non-heuristic alternatives is left for future work.

6.3 Enhancing The Robustness of Joint Recovery

Residual Analysis. We use the residual (loss function value) as a quality metric for the solution returned by the least-squares solver. A true solution should yield near-zero residual (up to numerical precision). If the residual exceeds a small threshold ϵ (e.g., 10^{-10}), we discard this solution and resample another set of N neurons. This process repeats until either the residual falls below ϵ or a maximum iteration limit T is reached, in which case we keep the solution with the smallest residual. Empirically, $T = 10$ suffices for all tested networks in Section 7.

Algorithm 1 Joint Neuron Signature Recovery in Layer $k+1$

Require: The number of selected neurons N; residual threshold ϵ; maximum iterations T.

Ensure: Neuron slopes $\tilde{s}_j^k, j \in \{1, \cdots, d_k\}$ in layer k, and neuron signatures $\tilde{A}_i^{k+1}, i \in \{1, \cdots, d_{k+1}\}$ in layer $k+1$.

Stage 1 (reconstruct affine spaces):

Reconstruct the affine space V_i for N_{tol} neurons in layer $k+1$. Without loss of generality, assume that the N_{tol} neurons are $\eta_i^{k+1}, i \in \{1, \cdots, N_{\text{tol}}\}$.

/* The joint recovery only requires that $N_{\text{tol}} > N$ and Stage 2 will not choose the same N neurons twice. In practice, we try to reconstruct affine spaces for all d_{k+1} neurons, i.e., $N_{\text{tol}} = d_{k+1}$. */

/* This step involves finding dual points, filtering points in deeper layers, and reconstructing affine spaces, which is directly done by the methods introduced in [5]. The dimension of reconstructed affine space should be $d_k + \text{rank}(A^k) - 1$, instead of $2d_k - 1$, which requires a minor adaptation of the consistency test in [5]; see Appendix B. */

Stage 2 (recover d_k neurons slopes):

For each affine space V_i where $i \in \{1, \cdots, N_{\text{tol}}\}$, randomly generate a set of basis vectors $\{U_0^i, \cdots, U_{m_k}^i\}$ of its orthogonal complement space $V_i^{\perp}$.

for $t \in \{1, \cdots, T\}$ **do**

  Randomly select N out of N_{tol} neurons, and denote by $\mathcal{B}$ the set of N indexs.

  Build the nonlinear system in Eq. (42), solve it for a solution and residual.

  /* Based on the solution, we get $\tilde{s}_j^k, j \in \{1, \cdots, d_k\}$ and $\overline{A}_i^{k+1}, i \in \mathcal{B}$ */

  if the residual does not exceed ϵ **then**

    Stop the loop, and go to Stage 3

  end if

end for

Choose $\tilde{s}_j^k, j \in \{1, \cdots, d_k\}$ corresponding to the solution with the smallest residual as recovered slopes.

Stage 3 (recover d_{k+1} neuron signatures):

Based on recovered neuron slopes, recover the d_{k+1} neuron signatures $\tilde{A}_i^{k+1}$ by the neuron signature recovery method in [5].

/*In this stage, the neuron splitting technique is not applied to layer k.*/

Return the recovered parameters $\tilde{s}_j^k, j \in \{1, \cdots, d_k\}$ and $\tilde{A}_i^{k+1}, i \in \{1, \cdots, d_{k+1}\}$.

Progressive Parameter Decoupling. The joint recovery method solves for neuron slopes and signatures simultaneously. However, once the neuron slopes $\widehat{s}_j^k$ are recovered, we can decouple the problem: with known neuron slopes, the neuron signatures $\tilde{A}_i^{k+1}$ can be recovered directly by the neuron signature recovery method in [5]. which is attributed to the following fact. Based on **Proposition** 1, we know that Z^k is not restricted, and one can recover the $(d_k - 1)$-dimensional subspace $\{Z^k | A_i^{k+1} Z^k + b_i^{k+1} = 0\}$, thereby allowing the method in [5] to work directly on Z^k.

6.4 Complexity Analysis

Complexity of The Joint Recovery Method. Algorithm 1 summarizes the joint neuron signature recovery method that adopts the two improvements, residual analysis and progressive parameter decoupling.

After adopting the two improvements, the joint recovery is divided into three stages as presented in Algorithm 1. Stages 1 and 3 directly adopt the methods in [5], which only require polynomial query and time complexity. As for Stage 2, we solve at most T non-linear systems, which requires polynomial time and no queries. Thus, the query complexity and time complexity of the joint neuron signature recovery method are both polynomial.

About persistent active/inactive neurons [12]. In [12], Ito et al. find that the probability of some neurons being active (or inactive) decreases exponentially with the network depth increases, which may make the complexity of the neuron signature recovery method in [5] degenerates into exponentially. We also perform the same test as [12] over PReLU FCNs. Concretely, we trained two PreLU FCNs with architectures of 784-32*8-10 (32*8 stands for 8 layers with 32 neurons in each layer) and 784-64*12-10, and collected 100,000 points for each model, and find that the neuron activation probability ranges from 0.2 to 0.8 and 0.1 to 0.9 respectively, which doesn't decrease exponentially with the network depth. Thus, it is expected that the complexity of our attack won't be signficantly affected.

7 Experiments

Practical experiments on numerous PReLU neural networks are performed to verify our work. Two types of networks are adopted in our experiments.

Random Networks. The first category comprises random networks, whose model parameters are initialized stochastically. Specifically, we adopt Kaiming uniform initialization, the default weight initialization scheme in PyTorch, to generate the model parameters. For the neuron-wise slopes, each is independently sampled from a uniform distribution over the interval $[0.1, 10]$, thereby breaking the slope restriction $s_j^k < 1$.

Real Networks Trained on MNIST. The second category is the real networks trained on MNIST, which is a typical benchmark dataset used in visual deep learning. It comprises ten classes of handwritten number gray images [16]. Each of the ten classes, i.e., '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9', contains 28×28 pixel gray images, totaling $60,000$ training and $10,000$ testing images. Based on MNIST, we train PReLU neural networks with typical model training settings (the loss is the cross-entropy loss; the optimizer is standard stochastic gradient descent; batch size 128). To ensure the presence of slopes greater than 1, we initialized the slope parameters uniformly from $[0.1, 10.0]$ and allowed them

to be updated during training. We observed that after training, about half of the slopes in each real network still exceed 1.

To compute the recovery error of model parameters, we first align the recovered model and the victim model to eliminate the influence of network isomorphism. Concrete details are provided in [6] and are not repeated here, as there are no changes. For convenience, we denote by $|\theta - \widehat{\theta}|$ the recovery error. For example, the notation $|s_j^k - \widehat{s}_j^k|$ stands for the recovery error of the slope s_j^k, although $\widehat{s}_j^k$ may equal to $\frac{1}{s_j^k}$ actually.

7.1 Verifying The Joint Neuron Signature Recovery

Effectiveness Verification To verify the effectiveness of the joint neuron signature recovery, we first collect 400 random networks and 400 real networks.

For random networks, four architectures are used: 30-35-40-10, 30-40-50-10, 25-30-35-10, and 30-35-40-45-10. For real networks trained on MNIST (downsampled to 14×14 to reduce the memory requirement and enable parallel testing), the adopted 4 architectures are 196-20-25-30-10, 196-20-30-40-10, 196-30-35-40-10, 196-25-30-35-10. Based on each architecture, we generate randomly or train 100 networks.

Next, the joint neuron signature recovery (i.e., Algorithm 1) is tested over these 800 networks. For each 2-deep (respectively, 3-deep) PReLU neural network, we apply Algorithm 1 to layer 2 (respectively, layer 3) to recover the neuron signatures in layer 2 (respectively, layer 3) and neuron slopes in layer 1 (respectively, layer 2). This subsection does not verify the influence of the progressive parameter decoupling strategy, therefore Stage 3 in Algorithm 1 is skipped.

Three findings are consistently observed in experiments on 800 networks:

1. All the experiments are finished within 10 iterations in Stage 2. In other words, a solution with a residual that does not exceed ϵ is returned.
2. The d_k neuron slopes are always recovered with high precision. Concretely, in all the experiments, the maximum recovery error (the maximum absolute difference between recovered slopes and the expected ones) of the neuron slopes does not exceed 2^{-10}. Fig. 4 summarizes the recovery error of neuron slopes on these 800 networks.
3. The precision of the recovered neuron signature, is lower than that of the recovered neuron slopes. Concretely, the recovery error $|A_i^{k+1} - \widehat{A}_i^{k+1}|$ of some neuron signatures exceeds 2^{-10} in some cases, although the residual does not exceed the threshold ϵ.

The second finding, in conjunction with the first, provides robust empirical validation of the effectiveness of the joint neuron signature recovery. The third finding is consistent with theoretical expectations, as our approach fundamentally relies on solving a nonlinear system. Besides, regarding the third finding, another possible explanation is as follows. In the non-linear system in Eq. (42), there are N equation constraints for each slope $\widehat{s}_j^k$, while only 1 equation constraint for each element in neuron signatures, making the precision of neuron

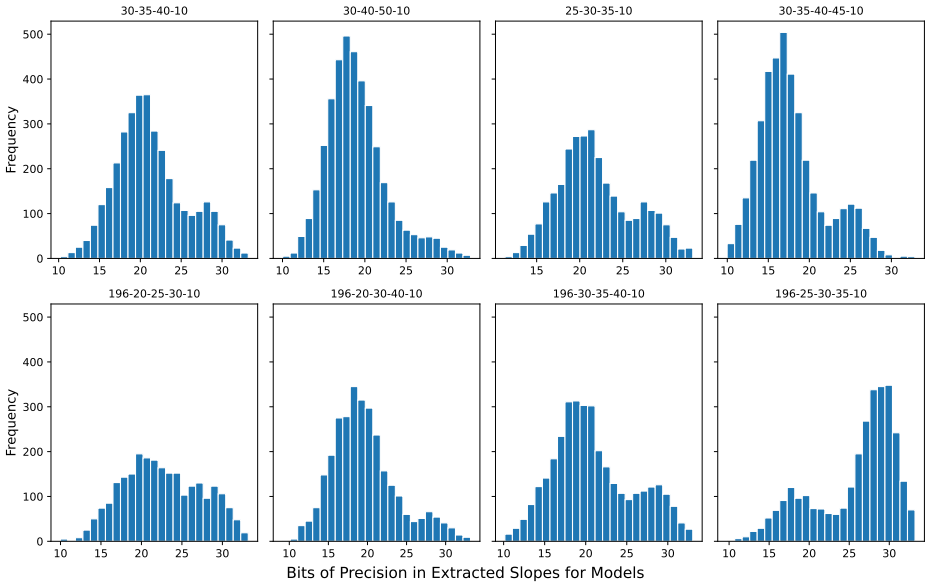

Fig. 4. Experiment results of the slope recovery over 800 PReLU networks.

slopes always high, and that of some neuron signatures low. Verification of this possible explanation is left to future work.

Running Time and Query Numbers. Overall, the running time and query numbers of our attack for a single model (recovering one layer) do not exceed 2^{11} seconds and 2^{26} queries, respectively. Taking the network 196-20-25-30-10 (recovering layer 3) as an example. Stage 1 in Algorithm 1 requires about $2^{9.645}$ seconds and $2^{24.93}$ queries on average. Stage 2 requires $2^{5.75}$ seconds on average and no queries. About $20,000$ dual points are searched in Stage 1, and many of which correspond to neurons in layer 2, instead of the target layer. Besides, we observe that the number of dual points used to reconstruct an affine space V_i is usually within 50.

Stability Verification In addition to effectiveness, the stability of the joint neuron signature recovery is also tested. Concretely, for a PReLU neural network used in the effectiveness verification, we run the effectiveness verification experiment 10 times. Each time new randomness seeds are used. Our focus is whether the d_k neuron slopes could be recovered with high precision each time.

For each architecture adopted in the effectiveness verification, we randomly select 1 from the 100 networks, and perform the above stability test. Experiments demonstrate that the d_k neuron slopes are consistently recovered with an recovery error smaller than 2^{-10}. Take one real network 196-20-25-30-10 as

an example. The 25 slopes in layer 2 are always recovered, and the maximum recovery error over 10 runs ranges from $2^{-25.90}$ to $2^{-13.07}$, with median $2^{-23.16}$.

7.2 Verifying The Progressive Parameter Decoupling Strategy

In this subsection, we test the influence of the progressive parameter decoupling strategy by performing the following experiment. Given a PReLU neural network f_θ, we apply the joint neuron signature recovery (i.e., Algorithm 1), and compare the recovery errors of neuron signatures obtained in Stage 2 and Stage 3 of Algorithm 1, respectively. Note that the progressive parameter decoupling strategy is applied in Stage 3.

Table 2 summarizes the experimental results on two networks, including a random network 20-22-24-26-10, and a real network 784-20-22-24-10 trained on MNIST. When the max recovery error (i.e., $\max|A_i^{k+1} - \widehat{A}_i^{k+1}|$) of a neuron signature is lower than 2^{-10}, we say that the neuron η_i^{k+1} is recovered successfully.

Table 2. Experimental results on the progressive parameter decoupling strategy. P.P.D.S.: Progressive Parameter Decoupling Strategy. η_{succ}: the ratio of successfully recovered neurons. $\max|A^k - \widehat{A}^k|$: the max error of successfully recovered signatures.

Setting	Architecture	Layer 2		Layer 3	
		$\max\|A^2 - \widehat{A}^2\|$	η_{succ}	$\max\|A^3 - \widehat{A}^3\|$	η_{succ}
Without P.P.D.S (Stage 2)	20-22-24-26-10	$2^{-11.20}$	21/24	$2^{-10.11}$	16/26
	784-20-22-24-10	$2^{-13.58}$	22/22	$2^{-13.77}$	22/24
With P.P.D.S (Stage 3)	20-22-24-26-10	$2^{-24.61}$	24/24	$2^{-17.64}$	26/26
	784-20-22-24-10	$2^{-27.98}$	22/22	$2^{-26.98}$	24/24

As shown in Table 2, some neurons are not recovered successfully in Stage 2. Take the network 20-22-24-26-10 as an example. There are 10 neuron signatures in layer 3 that are not recovered successfully. After adopting the progressive parameter decoupling strategy, all the neuron signatures are recovered successfully in Stage 3, and the maximum recovery error of successfully recovered neuron signatures also becomes much lower, thereby proving the positive influence of the progressive parameter decoupling strategy. Stage 3 reuses the dual points collected in Stage 1 and therefore does not incur additional queries.

7.3 More Exploration

Impact of Network Depth and Width At first, we test the influence of increased network depth and width on neuron slope recovery. We begin by constructing a PReLU network 30–5–6–7–8–9–10–11–12–10. Our method successfully recovers all neuron slopes in layers 1 through 7, achieving a maximum reconstruction error of $2^{-15.84}$. Subsequently, we evaluate a wide net-

work 100–110–120–128–10. Despite its large width, our approach remains effective—recovering all slopes with a maximum error of $2^{-9.85}$. In comparison with the results reported in Section 7.1, network depth exhibits no significant negative effect on recovery errors. Increased network width incurs a slight adverse impact, which may be attributed to the expansion of the nonlinear system.

About End-to-End Attack. We also make initial attempts at end-to-end attacks, but find them challenging within our current framework since we cannot refine the precision of our signature and slope solution. The precision refining is important for end-to-end attacks to avoid the failures caused by the accumulation of recovery errors in previous layers. We have considered the precision refining method in [6] that relies on solving a linear system, and found it infeasible for two reasons. First, it requires raw outputs to gather more critical points [6], while the raw output is not accessible in the hard-label setting. Second, for expansive PReLU networks, to refine the precision of the extended neuron signature in layer $k+1$, the rank of the coefficient matrix of the linear system should be $2d_k$. However, due to the linear constraints, the rank will always be smaller than $2d_k$. Thus, the precision refinement remains a challenge and is left to future work.

8 Conclusions

In this paper, we propose the first attack that works for expansive PReLU networks in the hard-label setting, without imposing any constraints on neuron slopes. This attack breaks the slope and structure restrictions of existing model parameter extraction attacks on PReLU neural networks. Experiments on several hundred networks (random and MNIST-trained) validate its effectiveness. Our technical contributions, especially the newly found network isomorphism (flip-and-scaling) and the new neuron signature recovery method, provide inspiration for future research on the cryptanalytic extraction of more complex neural networks. Besides, our work also raises some open problems for future research, such as highly optimized end-to-end attacks on expansive PReLU neural networks in the hard-label setting, and cryptanalytic extraction attacks against adversarially crafted neural networks.

A Proof And Empirical Verification of Proposition 1

A.1 Proof of Proposition 1

Proof. Let $d = d_i$. Assume for contradiction that there exists a non-zero vector $Q = [q_1, q_2, \cdots, q_d]^\top$ and a constant c such that $Q^\top Z^i = c$ holds for all feasible Z^i.

Under the assumption that the preactivation input of any neuron η^i_j will not always be greater or smaller than 0, we can find a network input X, ensuring that the preactivation input of η^i_j is 0. Then, consider another two inputs $X^+ =$

$X+\epsilon\cdot\Delta$ and $X^- = X-\epsilon\cdot\Delta$ (Δ is a random direction and ϵ is a small constant) which are located at the adajacent linear regions around X [8]. Then, according to the linear region properties of PReLU neural networks [8], for neurons η_l^i where $l \neq j$, their preactivation inputs (denoted by y_l^+ and y_l^-) corresponding to X^+ and X^- satisfy

$$y_l^+ \times y_l^- > 0,\ y_l^+ + y_l^- = 2y_l^i. \tag{44}$$

where y_l^i is the preactivation input given by X. For neuron η_j^i, we have $y_l^+ = -y_l^-$.

Next, consider the three post-activation outputs (denoted by Z^+, Z^-, and Z^i) corresponding to X^+, X^-, and X, we have $z_l^+ + z_l^- - 2z_l^i = 0$ for $l \neq j$, and $z_j^+ + z_j^- - 2z_j^i \neq 0$.

Since $Q^\top Z^+ = c$, and $Q^\top Z^- = c$, and $Q^\top Z^i = c$, one gets $Q^\top(Z^+ + Z^- - 2Z^i) = q_j \times (z_j^+ + z_j^- - 2z_j^i) = 0$, leading to the conclusion $q_j = 0$. Repeating for every $j \in \{1, 2, \cdots, d\}$ yields $Q = \mathbf{0}$, contradicting the assumption.

A.2 Empirical Verification of Proposition 1

To further support Proposition 1, we perform a simple numerical experiment on a typical expansive PReLU network with architecture 45-50-55-60-10. Network parameters are randomly initialized using Kaiming uniform distribution, and slopes are sampled uniformly from $[0.1, 10]$.

We sample 10^5 random inputs X and compute the post-activation outputs of the first hidden layer and the second hidden layer. These vectors are collected into two matrices: $M^1 \in \mathbb{R}^{10^5\times 50}$ and $M_2 \in \mathbb{R}^{10^5\times 55}$. We then compute the numerical rank of each matrix using SVD with a tolerance of 10^{-10}.

The results show that $\text{rank}(M_1) = 50$ and $\text{rank}(M_2) = 55$. Hence Z^1 spans the whole $\mathbb{R}^{50}$ and Z^2 spans $\mathbb{R}^{55}$. There is no linear constraints whatsoever on these post-activation outputs.

We repeat the same test on other architectures, including 30-20-25-30-10 and 20-25-20-15-10, and always obtain full rank. This empirical evidence strongly corroborates Proposition 1: the output Z^i is free, even when pre-activation inputs are subject to linear constraints (as happens in expansive layers).

B The Impact of Additional Linear Constraints on Affine Space Reconstruction

Due to the linear constraints, when filtering and clustering dual points using the IsConsistent Algorithm in [5], a minor change is applied. Concretely, In expansive PReLU neural networks (after splitting neurons), when two dual points correspond to the same neuron in layer $k+1$, the rank of the sum of two dual spaces is smaller than $d_k + \text{rank}(A^k) - 1$. In other cases, including two dual points from deeper layers or corresponding to different neurons, the resulting rank won't be smaller than $d_k + \text{rank}(A^k) - 1$.

C Converting Linear Constraints to Basis Vectors

Consider the m_k linear constraints $C^k H^k \overline{Z}^k = C^k B^k$ (see Eq. (39)) on $\overline{Z}^k$, the output of the extended layer k. As derived in Section 5.2, we have $C^k H^k = [C^k, C^k]$ where $C_k \in \mathbb{R}^{m_k \times d_k}$.

Therefore, the basis vectors U_l of the common subspace W (see Section 6.1) can be set to

$$\begin{aligned} U_l &= [u_{l,1}, u_{l,2}, \cdots, u_{l,2d_k},] \\ u_{l,j} &= u_{l,j+d_k} = C^k_{l,j}, \text{ for } l \in \{1, 2, \cdots, m_k\} \end{aligned} \tag{45}$$

where $C^k_{l,j}$ is the element located at row l and column j in matrix C^k. Note that $\{C^k_1, C^k_2, \cdots, C^k_{m_k}\}$ is a set of basis vectors of the left null space of A^k (see Section 5.2).

References

1. Baum, E.B.: A polynomial time algorithm that learns two hidden unit nets. Neural Comput. **2**(4), 510–522 (1990). https://doi.org/10.1162/NECO.1990.2.4.510, https://doi.org/10.1162/neco.1990.2.4.510
2. Baum, E.B.: Neural net algorithms that learn in polynomial time from examples and queries. IEEE Trans. Neural Networks **2**(1), 5–19 (1991). https://doi.org/10.1109/72.80287, https://doi.org/10.1109/72.80287
3. Blum, A., Rivest, R.L.: Training a 3-node neural network is np-complete. In: Hanson, S.J., Remmele, W., Rivest, R.L. (eds.) Machine Learning: From Theory to Applications - Cooperative Research at Siemens and MIT. Lecture Notes in Computer Science, vol. 661, pp. 9–28. Springer (1993). https://doi.org/10.1007/3-540-56483-7_20, https://doi.org/10.1007/3-540-56483-7_20
4. 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. Lecture Notes in Computer Science, vol. 16129, pp. 399–421. Springer (2025). https://doi.org/10.1007/978-3-032-06754-8_15, https://doi.org/10.1007/978-3-032-06754-8_15
5. 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. Lecture Notes in Computer Science, vol. 15601, pp. 364–396. Springer (2025). https://doi.org/10.1007/978-3-031-91107-1_13, https://doi.org/10.1007/978-3-031-91107-1_13
6. 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. Lecture Notes in Computer Science, vol. 12172, pp. 189–218. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_7, https://doi.org/10.1007/978-3-030-56877-1_7

7. 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. Lecture Notes in Computer Science, vol. 15491, pp. 207–236. Springer (2024). https://doi.org/10.1007/978-981-96-0944-4_7, https://doi.org/10.1007/978-981-96-0944-4_7
8. 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. Lecture Notes in Computer Science, vol. 16246, pp. 576–607. Springer (2025). https://doi.org/10.1007/978-981-95-5096-8_18, https://doi.org/10.1007/978-981-95-5096-8_18
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. 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, https://doi.org/10.1109/ICCV.2015.123
12. Ito, A., Miura, T., Todo, Y.: Is the hard-label cryptanalytic model extraction really polynomial? IACR Cryptol. ePrint Arch. p. 1868 (2025), https://eprint.iacr.org/2025/1868
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. Kurian, A., Aysu, A.: Train to defend: First defense against cryptanalytic neural network parameter extraction attacks. arXiv preprint arXiv:2509.16546 (2025)
15. Kurian, A., Aysu, A.: Train to defend: First defense against cryptanalytic neural network parameter extraction attacks. CoRR **abs/2509.16546** (2025). https://doi.org/10.48550/ARXIV.2509.16546, https://doi.org/10.48550/arXiv.2509.16546
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791, https://doi.org/10.1109/5.726791
17. Liu, H., Siproudhis, A., Experton, S., Lorenz, P., Boura, C., Peyrin, T.: Navigating the deep: Signature extraction on deep neural networks. CoRR

abs/2506.17047 (2025). https://doi.org/10.48550/ARXIV.2506.17047, https://doi.org/10.48550/arXiv.2506.17047

18. 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. Lecture Notes in Computer Science, vol. 14653, pp. 3–33. Springer (2024). https://doi.org/10.1007/978-3-031-58734-4_1, https://doi.org/10.1007/978-3-031-58734-4_1
19. 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, https://doi.org/10.1145/3287560.3287562
20. Qi, X., Lei, H., Wei, L., Sun, X., Wang, M.: Cryptanalytic extraction of neural networks with various activation functions. Cryptology ePrint Archive, Paper 2026/178 (2026), https://eprint.iacr.org/2026/178
21. Rivest, R.L.: Cryptography and machine learning. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings. Lecture Notes in Computer Science, vol. 739, pp. 427–439. Springer (1991). https://doi.org/10.1007/3-540-57332-1_36, https://doi.org/10.1007/3-540-57332-1_36
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. Sun, X., Lei, H., Wei, L., Qi, X., Hu, K., Wang, M., Wang, W.: Cryptanalytic extraction of convolutional neural networks. Cryptology ePrint Archive, Paper 2026/139 (2026), https://eprint.iacr.org/2026/139
24. 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