Polynomial Time Cryptanalytic Extraction of Deep Neural Networks in the Limited Architecture Knowledge Setting*

Yiqing Li and Gaoli Wang

Software Engineering Institute, East China Normal University, Shanghai, China
51275902064@stu.ecnu.edu.com, glwang@sei.ecnu.edu.cn

Abstract. Deep Neural Networks (DNNs) have emerged as a cornerstone of modern AI systems, rendering their internal parameters highly valuable intellectual property. Consequently, the security of DNNs against model extraction attacks has garnered significant research attention. Recent cryptanalytic extraction methods have demonstrated that recovering DNN parameters is feasible in polynomial time, both when attackers have exact logit access and in the more restrictive hard-label setting. However, a critical limitation of these state-of-the-art frameworks is the assumption that the attacker possesses a complete architecture knowledge of the target network, including the exact depth and width of the hidden layers. To date, the efficacy of parameter extraction algorithms under limited architecture knowledge remains unexplored.
This paper demonstrates that in the limited architecture knowledge scenario, existing cryptanalytic attacks suffer from neuron omission, which causes subsequent parameter recovery to fail entirely. To address this, we propose a general adaptive extraction workflow integrated with a novel error detection algorithm capable of identifying missing neuron anomalies and successfully recovering the correct parameters. Through experiment, we achieve the first successful extraction of a four-hidden-layer ReLU network (comprising over 1.1 million parameters) trained on CIFAR-10 under the limited architecture knowledge setting, while maintaining a 100% error detection recall.

Keywords: ReLU-Based Deep Neural Networks, Neural Network Extraction, Polynomial Query and Polynomial Time Attack, Limited Architecture Knowledge Attack.

1 Introduction

Deep neural networks (DNNs) have become a core engine driving advances in AI, achieving breakthrough progress in fields such as image recognition, natural language processing, and autonomous driving. By automatically learning complex features through hierarchical architectures, DNNs are now integrated

* This work was submitted to Crypto 2026 on Feb 12, 2026 and to ToSC 2026, Issue 3 on June 1, 2026.

into various technological developments, industrial manufacturing processes, and everyday applications.

The most common deep neural networks consist of a series of fully connected hidden layers. Each neuron within the network performs a global linear operation, followed by the parallel application of a local nonlinear operation. The parameters of a DNN are typically obtained by collecting large-scale training datasets—a process that can take months and cost millions of dollars, rendering the trained DNN a highly valuable asset. However, these assets are often made available via oracle interfaces $\mathcal{O}$, allowing arbitrary users to submit inputs to a DNN classifier and receive the corresponding outputs.

Over the past several years, a substantial body of research has focused on extracting DNN parameters through such oracle access. And significant progress has been made in the field of model extraction for DNNs [5–7,11–13,16], with the goal of recovering the parameters of each individual neuron with high precision.

In 2020, Carlini et al. [3] highlighted the structural resemblance between DNNs and block ciphers: both are composed of alternating layers of global linear operations and local nonlinear operations—ReLU in DNNs and S-boxes in block ciphers. State-of-the-art model extraction attacks have since drawn upon concepts from differential cryptanalysis of block ciphers.

In the study by Carlini et al. on cryptanalytic extraction of neural network models [3], the authors formulated this problem under several assumptions regarding the oracle $\mathcal{O}$ and the attacker's knowledge:

- **Architecture knowledge.** Knowledge of the neural network architecture is required.
- **Full-domain inputs.** Arbitrary inputs from $\mathcal{X}$ can be submitted.
- **Output Settings.** Raw outputs are received directly from the model f without further processing, while hard-label outputs are received from the final predicted class label (e.g., the most likely class), without any confidence information. This paper focuses primarily on the logit-output setting.
- **Precise computations.** The function f is specified and evaluated using 64-bit floating-point arithmetic.
- **Scalar outputs.** Without loss of generality, the output dimensionality is assumed to be 1, i.e., $\mathcal{Y} = \mathbb{R}$.
- **ReLU Activations.** All activation functions (σ's) are ReLUs.

The authors employed several cryptanalytic techniques to recover neuron weights and biases with high precision in polynomial time, a process referred to as *neuron signature recovery*. However, since neuron weights are typically recovered up to an unknown multiplicative factor, the sign of this factor directly affects the fidelity of the overall model recovery. The aforementioned work resorts to a nearly exhaustive search over the correct signs, incurring exponential time complexity. This limitation was later addressed in [1], where the authors introduced the *Neuron Wiggle* technique, enabling sign recovery in polynomial time. With this advancement, parameter recovery in the logit-output setting has reached a mature stage.

The paper [10] systematically investigates the limitations of existing neural network model extraction methods in deeper architectures. They identify two critical bottlenecks in the signature extraction process when applied to deep ReLU networks: rank deficiency in the linear systems, which leads to incorrect signature recovery, and the unintended inclusion of critical points from deeper layers, which are mistakenly merged into components of the target layer. To address these issues, the authors propose a subspace intersection method to increase the rank of the system, a filtering mechanism to discard deeper critical points, and a hybrid sign recovery strategy that combines the system-of-equations approach with the neuron wiggle method to achieve high-confidence sign determination. Based on these improvements, [10] presents the first end-to-end polynomial-time extraction of eight-layer ReLU networks.

Moreover, Carlini et al. [3] also identified the relaxation of these assumptions as a direction for future work. Among these, the Output Settings assumption has seen recent progress in the more challenging hard-label setting, as explored in [2, 4].

Subsequent research has shifted toward model extraction under different output settings. The work of [12] defines five types of attack scenarios: (S1) the most likely class label (referred to as the hard-label setting), (S2) the most likely class label along with its probability score, (S3) the top k labels and their probability scores, (S4) all labels and their probability scores, and (S5) the raw output of the DNN. Prior black-box DNN attacks have predominantly focused on the simplest scenario (S5), requiring only a polynomial number of queries and polynomial time [1]. Chen et al. [4] made initial progress in the hard-label setting by recovering signatures in a limited manner using information derived from decision boundary points.

These challenges were later resolved in [2]. Building upon the foundation of [1], that work introduced a series of optimizations tailored to the hard-label setting. First, dual points—which simultaneously serve as decision boundary points and critical points—are employed to extract information from the model. Singular Value Decomposition (SVD) is then used to cluster dual points corresponding to individual neurons, thereby recovering their signatures. Finally, in contrast to the rate-of-change computation employed in [1], the authors achieve sign recovery via a distance-based method.

Meanwhile, some researchers began to focus their attention on other commonly used neural network models, such as Convolutional Neural Networks (CNNs), whose unique structural characteristics pose a number of challenges. In a subsequent work by Sun et al. [15], the first attack framework capable of systematically and fully extracting the parameters of Convolutional Neural Networks (CNNs) under the hard-label setting was proposed. Its high precision and practical applicability were demonstrated on several CNN architectures, including LeNet-5. The paper [9] studies model extraction of CNNs with ReLU and max-pooling in the soft-label setting. They generalize critical points to max-pooling, showing they reveal relative information $|\eta_a - \eta_b|$ rather than absolute comparisons with zero. They then exploit the staircase sparsity of convolution to

recover convolutional rows without sign ambiguity. Furthermore, they introduce receptive-field analysis to identify the layer of critical points, enabling systematic noise filtering. Experiments on a VGG-style CNN trained on CIFAR-10 validate their approach.

Despite these advances, existing research on DNN extraction universally assumes complete knowledge of the network architecture. The problem of DNN extraction under the limited architecture knowledge setting remains largely unexplored. This paper aims to advance the relaxation of the *Architecture knowledge* assumption. Thus, we propose a novel assumption:

– **Limited architecture knowledge.** Knowledge of the neural network architecture is limited. Specifically, we cannot use knowledge of model architecture such as the number of hidden layers and the number of neurons in each layer. This significantly affects the existing parameter extraction attacks.

Given only a set of input-output pairs, the attacker can accurately and efficiently extract the parameters of deep neural networks. Eventually, we succeeded in extracting the neural network parameters from the model with the architecture 3072-256×3-64-10. These works, along with prior work on DNN model parameter extraction, are shown in Table 1.

Table 1: **Comparison of Neural Network Extraction Methods.** Previous works successfully extracted either single-hidden-layer models or networks with very few hidden neurons, requiring full Knowledge of the neural network architecture or often resulting in exponential time costs. In contrast, our new attack demonstrates the ability to extract a multilayer neural network with approximately 1,600 hidden neurons only requiring limited knowledge of the neural network architecture

	Neural Network				**Extraction**
Architecture	Hard-Label	LAK	Signature	Sign	**Approach**
10-20-20-1	×	×	*poly*	*exp*	ICML'20 [14]
10-20-20-1	×	×	*poly*	*exp*	Crypto'20 [3]
784-128-1	×	×	*poly*	*exp*	Crypto'20 [3]
3072-256×8-10	×	×	*poly*	*poly*	EC'24 [1]
100-50-50-1	×	×	*poly*	*poly*	[6]
3072-16×8-10	×	×	*poly*	*poly*	EC'26 [10]
1024-2-2-1	✓	×	*restr.*	*exp*	AC'24 [4]
3072-256×3-64-10	✓	×	*poly*	*poly*	EC'25 [2]
3072-512×3-64-10	×	✓	*poly*	*poly*	**This work**

Hard-Label:×: Full access to output logits. ✓: Restricted hard-label scenario.
LAK(Limited Architecture Knowledge):×: Full knowledge of the neural network architecture. ✓: Limited knowledge of the neural network architecture.
poly: Polynomial time. *exp*: Exponential time. *restr.*: Restricted to certain architectures.

1.1 Why Previous Techniques Cannot Be Used

To understand why state-of-the-art model extraction frameworks fail under limited architecture knowledge, it is essential to examine their underlying operational logic. Existing cryptanalytic extraction algorithms, spanning from the foundational work of Carlini et al. [3] to recent extensions [1, 10], are fundamentally engineered as sequential, layer-by-layer reconstruction processes. The mathematical formulation for extracting the parameters of the $i+1$-th hidden layer strictly presumes that the weights and biases of the i-th layer have already been reconstructed to a high degree of numerical precision. Any micro-scale perturbation or omission in the preceding layer inevitably invalidates the algebraic preconditions required by the subsequent stages.

Under the standard assumption of complete architecture knowledge, the attacker is privy to the exact dimension d_i (the number of hidden neurons) of the target layer i. This architectural prior serves as a decisive stopping criterion during the signature collection phase. Specifically, the attack algorithm continuously samples input queries to trigger and isolate the critical points associated with the i-th layer until exactly d_i unique neuron signatures are resolved.

However, when operating under the limited architecture knowledge setting, this stopping criterion vanishes. And the layer-wise distribution remains a black box. Consequently, the signature recovery process lacks a localized termination signal. The extraction algorithm cannot deterministically verify whether it has exhausted all hidden neurons within the current layer. If the sampling density is insufficient or if certain neurons possess narrow activation regions, the algorithm will prematurely proceed to the next layer, resulting in what we refer to as neuron omission.

The algebraic consequences of omitting even a single neuron are catastrophic for subsequent parameter recovery. Mathematically, the parameters extracted from the i-th layer define the basis of the affine space that projects inputs into the activation environment of the $i+1$-th layer. When a neuron is omitted, the computed transformation matrix suffers from a dimensional deficiency. As a result, when the attacker attempts to collect critical points to reconstruct the $i+1$-th layer, the gathered algebraic relations no longer align with the expected hyperplanes. Instead, they manifest as numerical noise.

1.2 Our Contributions

The main contribution of this paper lies in proposing a novel method for extracting target model parameters under the limited architecture knowledge setting. Section 2 provides a concise overview of the preliminaries, including core concepts and existing parameter extraction techniques. Section 3 introduces our general adaptive extraction workflow, analyzes the influence of missing neurons on subsequent parameter recovery, and details our novel error detection algorithm. Section 4 presents the experimental setup, defines the evaluation metrics, and provides empirical results validating the effectiveness of our framework. Finally, Section 5 concludes the paper and discusses open problems for future research.

Thus, our main contributions are as follows:

- **A General Adaptive Extraction Workflow.** This work presents the first study on the extraction of Deep Neural Networks under the limited architecture knowledge setting. We systematically analyze the severe impact this assumption imposes on existing signature and sign recovery algorithms [1,3], demonstrating how neuron omissions lead to cascading failures in deeper layers. To address these challenges, we propose a general and adaptive extraction workflow capable of dynamically determining both the network's depth and the number of neurons per hidden layer without requiring prior architectural specifications.
- **Efficient and Accurate Error Detection.** Building upon the proposed adaptive workflow, we introduce a highly efficient and accurate error detection algorithm designed specifically to pinpoint missing neuron anomalies. By leveraging the algebraic properties of dimensional deficiency, this algorithm efficiently halts the propagation of extraction errors before they affect subsequent layers. We demonstrate that integrating this detection mechanism into our general workflow significantly enhances the robustness of foundational cryptanalytic extraction techniques [3], ensuring reliable parameter recovery under limited architecture knowledge setting.

It is particularly worth noting that Liu et al. [10] identify two key obstacles in deep ReLU network extraction: rank deficiency in signature recovery and interference from deeper-layer critical points, and propose corresponding solutions. We also briefly discuss the effects of interference from neurons in subsequent layers in Section 3.4. Furthermore, extending our error detection mechanism to the hard-label scenario presents a distinct challenge: without access to exact output logits, it is highly difficult to detect the dimensional deficiency of the preceding layer using the same algebraic consistency checks.

Our code is publicly available at https://github.com/CryptoAiLi/DNNExtractionLAK.

2 Preliminaries

In this chapter, we first introduce important definitions and notation (Section 2.1). For the remaining content, please refer to Appendix A.1. Next, we describe the attack goal and workflow of model parameter extraction (Section 2.2). Finally, we elaborate on the algorithms employed in prior work [1] (Section 2.3).

2.1 Definitions and Notation

Here we present some basic definitions and notations used throughout the manuscript which closely follows the terminology first presented in [3] and then adopted in paper [1, 2, 4].

Definition 1 (r-Deep Neural Network). *An r-deep fully connected neural network f_θ is a function parameterized by θ, which maps inputs from a vector*

space $\mathcal{X} = \mathbb{R}^{d_0}$ to an output space $\mathcal{Y} = \mathbb{R}^{d_{r+1}}$. The architecture of the network is determined by a sequence of dimensions $[d_0, \ldots, d_{r+1}]$, and the function f_θ is formulated as a composition of alternating linear transformations f_i and a component-wise non-linear activation function σ:

$$f_\theta = f_{r+1} \circ \sigma \circ \cdots \circ \sigma \circ f_2 \circ \sigma \circ f_1$$

where each linear operation evaluates as $f_i(x) = A^i x + b^i$. The fundamental notations used to describe this architecture are specified as follows:

- *r: The total number of hidden layers, representing the depth of the neural network.*
- *d_i: The dimensional width of the i-th layer, corresponding to the number of constituent neurons.*
- *f_i: The affine transformation mapping of the i-th layer, mapping $\mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$.*
- *A^i: The weight matrix governing the i-th layer, where $A^i \in \mathbb{R}^{d_i \times d_{i-1}}$.*
- *b^i: The bias vector associated with the i-th layer, where $b^i \in \mathbb{R}^{d_i}$.*
- *θ: The comprehensive set of network parameters, encompassing all layer weights A^i and biases b^i.*

As in [1–4], we only consider neural networks using the ReLU activation function defined as $\sigma : x \mapsto \max(x, 0)$.

Definition 2 (Neuron Value and State). *Let $\mathcal{V}(\eta; x)$ denote the value that neuron η takes with $x \in \mathcal{X}$ before applying σ. If $\mathcal{V}(\eta; x) > 0$ then η is* active. *Otherwise, the neuron is* inactive. *The state of η on input x (i.e., active, or inactive) is denoted by $\mathcal{S}(\eta; x)$.*

Definition 3 (Critical Point). *A* critical point *x satisfies $\mathcal{V}(\eta; x) = 0$ for some neuron η the value $\mathcal{V}(\eta; x) = 0$.*

Definition 4 (Linear Neighborhood).
Let $x \in \mathcal{X}$. The linear neighbourhood *of x is the set*

$$\{u \in \mathcal{X} \mid \mathcal{S}(\eta; x) = \mathcal{S}(\eta; u) \text{ for all neurons } \eta \text{ in the DNN}\}.$$

Since the state of all neurons remains the same for all inputs in the same linear neighbourhood, the DNN behaves as a linear map (within that linear neighbourhood). Let $F_{i,j} = \sigma \circ f_j \circ \cdots \circ \sigma \circ f_i$, $1 \le i \le j \le r$. Then, for all x in a linear neighbourhood,

$$\begin{aligned} F_{i,j}(x) &= I^j(A^j \cdots (I^{i+1}(A^{i+1}(I^i(A^i x + b^i)) + b^{i+1}) \cdots + b^j) \\ &= Ix + \beta, \end{aligned}$$

where I^ℓ are $0-1$ diagonal matrices with a 0 on the diagonal's k-th entry when neuron k at layer ℓ is inactive and 1 when that neuron is active.

Let $i \in \{1, \ldots, r+1\}$. The DNN can be regarded as a composition of an *input function* F^{i-1}, the i-th layer and an *output function* G^{i+1}:

$$f = \underbrace{f_{r+1} \circ \sigma \circ \cdots \circ \sigma \circ f_{i+1}}_{G^{i+1}} \circ \sigma \circ f_i \circ \underbrace{\sigma \circ f_{i-1} \circ \cdots \circ \sigma \circ f_1}_{F^{i-1}}.$$

When $i = 1$, F_0 is the identity map on the input; when $i = r+1$, G_{r+2} is the identity map on the output. Furthermore, if we restrict inputs x' to be in the linear neighbourhood of $x \in \mathcal{X}$, we can write F^{i-1} and G^{i+1}, respectively, as

$$F^{i-1}(x') = F_x^{i-1}x' + b_x^{i-1} \quad \text{and} \quad G^{i+1}(x') = G_x^{i+1}x' + b_x^{i+1}.$$

In the context of our attack, i is the index of the target layer and F^{i-1}, G^{i+1} represent, respectively, the recovered and non-recovered layers of the network.

2.2 Attack Overview

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

Adversarial goal. The objective of the extraction is not to exactly replicate the target network, but rather to achieve what is known as an (ϵ, δ)-functionally equivalent extraction [3].

Definition 5 (functionally equivalent extraction). *We say that two models f and $\hat{f}$ are (ϵ, δ)-functionally equivalent on an input space $S \subset \mathbb{R}^d$ if $\forall x \in S, \mathbb{P}(|\frac{\hat{f}(x)-f(x)}{f(x)}| \leq \epsilon) \geq 1 - \delta$.*

Following the general pipeline established in prior work [3], our attack proceeds in a layer-wise manner to extract the model parameters θ. The analysis of each layer consists of three key steps:

- **Error Detection:** This step checks whether any neurons in the previous hidden layer are missing. A detailed explanation of why the verification of the preceding layer is performed at the beginning of each layer's parameter extraction is provided in Section 3.
- **Signature Recovery:** In this step, the parameters of each neuron are recovered individually. It is important to note that the recovered parameters are typically scaled by an unknown multiplicative factor. Essentially, this step determines the position of the critical hyperplane.
- **Sign Recovery:** This step determines the sign of the unknown multiplicative factor, ensuring that the input can propagate through the neuron without altering its activation state. In essence, this step identifies which side of the critical hyperplane corresponds to the active region.

During the parameter extraction process, the recovered weight column vector of a neuron is typically scaled by an unknown multiplicative factor. We refer to this computed approximate weight column vector as the signature.

Definition 6 (Neuron Signature). *The signature $\tilde{A}^i_\eta$ of a neuron η is defined as $\alpha \cdot A^i_\eta$, where α is an arbitrary scaling factor applied to its parameters.*

Our terminology closely follows that originally introduced in [3] and adopted by recent attacks [1, 2, 4]. The primary distinction from prior attacks lies in the absence of neural network architecture information. We cannot ascertain whether an entire layer has been correctly recovered, which evidently introduces numerous complications for subsequent neuron recovery.

In the actual neural network recovery process, due to issues with critical point sampling, it is likely that a given neuron cannot obtain at least two critical points, making it impossible to identify the neuron's existence during clustering. Naturally, one might consider solutions such as sampling a sufficient number of dual points. However, deeper neurons are more difficult to acquire, and the critical points corresponding to deeper neurons may introduce noise into the clustering of critical points for the current layer. Therefore, simply increasing the number of critical points does not adequately solve the problem. First, we need an error detection method that can accurately identify potential missing neurons in the previous layer. This enables us to re-execute the parameter extraction for that layer and prevent errors from affecting the recovery of subsequent neuron parameters.

Second, due to the unknown depth of the network architecture, we also require a decision logic to accurately determine when the extraction of all neural network model parameters is complete.

In summary, our new neural network parameter extraction process improves upon the original algorithm in two main aspects:

- Before proceeding with layer-by-layer signature recovery and sign recovery, we check the recovery status of the previous hidden layer.
- We incorporate corresponding decision algorithms into the overall workflow to ensure accurate determination and recovery of the neural network depth.

In Section 3, we detail the new general parameter extraction workflow under these improvements; furthermore, we also prominently introduce our brand-new Error Detection algorithm.

2.3 Overview of the Existing Parameter Extraction in the logit-output setting

In this section, we formally review how state-of-the-art cryptanalytic extraction algorithms operate in the logit-output setting. The core of the signature recovery algorithm, as proposed by Carlini et al. [3] and subsequently refined by Canales-Martínez et al. [1], relies on differential cryptanalysis techniques.

We first formally define the neural network function f_θ according to Definition 1. Given an input X, the output logits $f_\theta(X)$ is computed as

$$f_\theta(X) = A^{r+1} \cdot \left(A^2 I^1 \left(A^1 X + b^1\right) + b^2\right) \cdots + b^{r+1}. \tag{2.1}$$

where I^i are $0-1$ diagonal matrices with a 0 on the diagonal's k-th entry when neuron k at layer ℓ is inactive and 1 when that neuron is active.

Carlini et al. exploit the abrupt change in the state of a neuron at a critical point (Definition 3), which leaks the weight information of that neuron, to compute and obtain the signature of the neuron.

Finding Critical Points. Let $x_1, x_2 \in \mathbb{R}^{d_0}$ and $x^* \in (x_1, x_2)$. And the function $f(x^*)$ is piecewise linear (see definition 4), and each discontinuity along the line segment from x_1 to x_2 corresponds to a critical point. These critical points can be readily located using binary search.

However, at this stage, it is not yet known which neuron corresponds to each critical point. Therefore, before proceeding with layer-wise parameter recovery, we first identify the set of critical points belonging to the target layer i via critical point clustering. In brief, for critical points associated with neurons in the same target layer, the signatures $\tilde{\mathbf{A}}_\eta^i$ recovered by the signature recovery algorithm described in the next paragraph should be parallel. The detailed clustering algorithm is described in [3].

Signature Recovery. Without loss of generality, consider the recovery of the first neuron η_1^i in layer $i \in \{2, \ldots, n\}$, and assume that the first $i-1$ layers have been recovered.

Let x be a critical point corresponding to the neuron η_1^i, and denote by $y \in \mathbb{R}^{d_{i-1}}$ the resulting input of the i-th fully connected layer (i.e., the output of the $(i-1)$-th activation layer).

As defined in Definition 4, the critical point x has two linear neighborhoods, denoted as ℓ_+ and ℓ_-, on either side. The corresponding neuron activation state matrices I^i differ only in the activation state of the first neuron η_1^i, while the activation states of all other neurons remain identical.

Consider a set of basis vectors $H \subseteq \mathbb{R}^{d_{i-1}}$[1] and perturb y along the directions H and $-H$ with a small step size ϵ. The resulting points $y + \epsilon H$ and $y - \epsilon H$ remain within the two respective linear neighborhoods. Assuming $y + \epsilon H \in \ell_+$ and $y - \epsilon H \in \ell_-$, we obtain:

$$\begin{aligned} f_\theta(y) &= G^{i+1} I^i (A^i y + b^i) + B^{i+1}, \\ f_\theta(y + \epsilon H) &= G^{i+1} I_+^i (A^i (y + \epsilon H) + b^i) + B^{i+1}, \\ f_\theta(y - \epsilon H) &= G^{i+1} I_-^i (A^i (y - \epsilon H) + b^i) + B^{i+1}, \end{aligned} \tag{2.2}$$

where $G^{i+1} = [g_1^{i+1}, \cdots, g_{d_i}^{i+1}] \in \mathbb{R}^{d_i}$ and $b^{i+1} \in \mathbb{R}$ are determined by the critical point x.

Due to the properties of linear neighborhoods, the matrices I^i, I_+^i, and I_-^i in Equation (4.2) differ only in the activation state of neuron η_1^i. Hence, we can recover $|\alpha|$ via a differential attack as follows:

[1] In practice, such a set H can be obtained by randomly sampling several vectors in the input space $\mathcal{X}$ and propagating them through the first $i-1$ hidden layers.

$$\begin{aligned} \alpha_+ &= f_\theta(y + \epsilon H) - f_\theta(y), \\ \alpha_- &= f_\theta(y) - f_\theta(y - \epsilon H), \\ |\alpha| &= |\alpha_+ - \alpha_-| = g_1^{i+1} A_1^i H. \end{aligned} \tag{2.3}$$

Since we assume the output dimension is 1, g_1^{i+1} remains constant within both linear neighborhoods. By choosing a sufficiently large set of directions $H = \{H_j\}$, we can construct a system of linear equations and solve for the signature $\tilde{A_1^i}$. In the signature recovery process, the sign of $A_1^i H_j$ cannot be directly obtained. To overcome this obstacle, the proposed algorithm first assumes $A_1^i H_1 > 0$ and subsequently recovers the signs corresponding to other directions H_j under this premise. By constructing the combined direction $H_1 + H_j$ and computing its second-order difference α_{1+j}, and then comparing the absolute differences between $|\alpha_{1+j}|$ and $|\alpha_1 + \alpha_j|$ versus $|\alpha_{1+j}|$ and $|\alpha_1 - \alpha_j|$, the sign of $(A_1^i H_1)(A_1^i H_j)$ can be determined. This enables the inference of the sign of $A_1^i H_j$.

Sign Recovery. After recovering the signature of each neuron in layer i (i.e., the ratios of its weights), the remaining task is to determine the sign of each neuron. Recall that multiplying all weights and the bias of a neuron by -1 preserves its signature but flips the behavior of its ReLU: the neuron becomes active where it was previously inactive, and vice versa. Thus, without the correct sign, we cannot peel off the layer. Let $\hat{A}_j^i$ be the recovered signature (a row vector) for the j-th neuron in layer i, and let $s_j \in \{+1, -1\}$ be the unknown sign. The actual weights are $A_j^i = s_j \cdot \hat{A}_j^i$, and similarly for the bias. We propose three polynomial-time techniques to recover all signs s_j, avoiding the exponential search required by prior work [3].

- **SOE (System of Equations).** When the number of degrees of freedom at layer i is at least d_i, we can recover all signs simultaneously by solving a single linear system. For a random input x in the linear neighborhood of a chosen point, we query $f(x)$ and $f(x + \Delta_k)$ for d_i carefully chosen perturbations Δ_k. From the output differences we obtain a system $c \cdot y_k = z_k$ where c contains the unknown signs. Solving the system reveals which neurons are inactive around x, hence determining the signs.
- **Neuron Wiggle.** When the degrees of freedom are insufficient for SOE, we use a statistical method. For a target neuron j, we find a critical point x^* and construct a wiggle Δ in the input space that maximally changes the input to neuron j while minimally affecting others. We then query $f(x^* - \Delta)$ and $f(x^* + \Delta)$. If the neuron has positive sign, the large change propagates to the output on the $+\Delta$ side; if negative, it propagates on the $-\Delta$ side. By comparing the magnitudes $|f(x^* + \Delta) - f(x^*)|$ and $|f(x^* - \Delta) - f(x^*)|$ across multiple critical points (e.g., $s = 200$), we gather statistical evidence to determine the sign with high confidence.

Last Layer Recovery. Unlike hidden layers, the output layer (layer $r + 1$) contains no ReLU activation and therefore has no critical points. This absence can be detected during critical point clustering: if no critical point is found for a given layer, we identify it as the output layer.

Let $y = F_r(x) \in \mathbb{R}^{d_r}$ denote the output of the last hidden layer (i.e., after the ReLU of layer r). For any input x, the final network output is given by an affine transformation:

$$f_\theta(x) = A^{r+1}y + b^{r+1},$$

where $A^{r+1} \in \mathbb{R}^{d_{r+1} \times d_r}$ and $b^{r+1} \in \mathbb{R}^{d_{r+1}}$.

Since we have already recovered all previous layers, we can compute $y = F_r(x)$ for any chosen input x. By collecting $m \geq d_r + 1$ distinct inputs $\{x_k\}_{k=1}^m$ and their corresponding outputs $f_\theta(x_k)$, we construct the linear system:

$$\begin{bmatrix} F_r(x_1)^T & 1 \\ \vdots & \vdots \\ F_r(x_m)^T & 1 \end{bmatrix} \begin{bmatrix} (A^{r+1})^T \\ (b^{r+1})^T \end{bmatrix} = \begin{bmatrix} f_\theta(x_1)^T \\ \vdots \\ f_\theta(x_m)^T \end{bmatrix}.$$

This system is solved via ordinary least squares to obtain $\tilde{A}^{r+1}$ and $\tilde{b}^{r+1}$, completing the full parameter extraction.

The Architectural Bottleneck. The mathematical elegance of this pipeline fundamentally collapses when the layer dimension d_i is unknown. The critical point clustering phase relies on d_i as an absolute stopping condition. Without this prior, the signature recovery phase will prematurely halt, leading to neuron omissions. Consequently, the subsequent sign recovery phase—which assumes the mathematical completeness of the recovered signatures—will execute over a corrupted affine basis, cascading into a catastrophic rank deficiency for all subsequent layers.

3 Improving Parameter Extraction

In the limited architecture knowledge scenario, the signature recovery algorithms for logit-output settings remain effective, because the recovery process for each neuron within the same layer is independent of one another. At the same time, this also means that during the recovery of neuron signatures in the target layer, errors in other neurons of the same layer cannot be detected. Therefore, we can only, and need to, attempt to detect such omissions before the parameter recovery of subsequent layers. Missing neurons introduce certain inconsistencies during the parameter extraction of the next layer, which can be leveraged to verify the recovery status of the preceding layer.

This section focuses on characterizing how such inconsistencies arise and what impact they have on subsequent neuron recovery, and proposes our adaptive extraction attack pipeline and error detection mechanism. In section 3.1, we first discuss the specific impact that missing neurons have on the recovery of subsequent neurons. Subsequently, section 3.2 outlines our general adaptive parameter extraction workflow designed for structural uncertainty. Section 3.3 details our novel error detection algorithm tailored specifically for the logit-output setting. Finally, Section 3.4 discusses noise filtering and the conditions under which redundant neurons do not affect subsequent recovery.

3.1 Influence on Parameter Extraction

In cryptanalytic extraction frameworks, the initial phase of critical point collection and the subsequent recovery of neuron parameters within the same layer typically operate independently of the network's internal layout. Specifically, locating critical points along line segments via binary search remains completely unaffected by the limited architecture knowledge setting, as it solely relies on observing the piecewise linear discontinuities of the oracle's black-box outputs. Consequently, the parameter extraction algorithm itself remains functionally effective for the currently targeted layer even if some neurons are missed. However, when operating under the limited architecture knowledge setting, undetected missing neurons in the previous layer exert profound cascading effects on the subsequent layer's recovery, affecting the signature and sign recovery phases differently.

Influence on Signature Recovery. Taking the signature recovery algorithm in the logit-output setting as an example, if a neuron is omitted in layer $i-1$, the affine space mapping to layer i becomes distorted. Specifically, approximately 50% of the sampled critical points—those where the missing neuron would have been active—will manifest as irreconcilable numeric noise. As we will detail in the error detection mechanics (Section 3.2), only the 50% of critical points where the missing neuron remains inactive can be successfully clustered and utilized for subsequent parameter extraction. This drastic reduction in valid critical points inherently leads to decreased precision, potential rank deficiency, and catastrophic failure in the signature recovery of subsequent neurons.

In addition to neuron omission, signature recovery is affected by noise from subsequent layers. As discussed in [10], interference from deeper neurons can produce critical point components that do not belong to the target hidden layer. The noise components observed in Section 4.2, Table 2, also demonstrate that non-target components occur in practice. Accepting such components as target neurons introduces neuron redundancy. We discuss noise filtering and the treatment of redundant neurons in Section 3.4. **Influence on Sign Recovery.** At first glance, the sign recovery phase appears relatively resilient to intra-layer omissions. Because algorithms like *Neuron Wiggle* [1] evaluate each resolved signature entirely independently using localized input perturbations, the sign recovery for successfully extracted neurons within the current layer can often execute without realizing that the layer's global architecture is incomplete.

However, this undetected absence in a previous layer will inevitably cause unpredictable and fatal interference in the sign recovery of subsequent layers. A core mathematical step of the *Neuron Wiggle* algorithm relies on observing the downstream impacts of targeted perturbations to accurately distinguish the critical hyperplanes of prior layers from those of subsequent layers. When a neuron is missing from an earlier layer, the resulting distortion in the affine projection severely confounds this distinction, introducing substantial noise that directly interferes with accurate sign determination.

Therefore, the hidden structural anomaly of omissions not only triggers catastrophic rank deficiencies during subsequent signature recovery but also funda-

mentally corrupts subsequent sign recovery. This cascading cross-layer vulnerability definitively underscores the absolute necessity of implementing an early error detection mechanism before advancing to any downstream stages.

3.2 Adaptive Parameter Extraction Workflow

This subsection presents our general adaptive model parameter extraction workflow designed for the limited architecture knowledge setting. We first discuss the theoretical baseline of how extraction proceeds under structural uncertainty, followed by our optimized workflow that incorporates an efficient early-warning mechanism.

Initialization. Collect an initial large pool of critical points P using the critical point finding algorithm employed in prior attacks. We only require a rough estimate of the total scale of the network N (total number of neurons) and sample $\mathcal{O}(N \log N)$ critical points accordingly.

Layer-wise Recovery and The Baseline Inefficiency. In a general parameter recovery workflow lacking a specialized error detection mechanism, the number of clustered critical points dictates the number of recovered neurons in layer i. The algorithm sequentially applies signature and sign recovery. It is worth noting that even if a neuron was omitted in layer $i-1$, the parameter extraction for layer i can theoretically proceed in a highly inefficient manner. Specifically, critical points where the missing neuron remains *inactive* are still mathematically valid and can yield correct parameters. However, approximately 50% of the critical points (where the missing neuron is active) will manifest as numeric noise and be discarded. This triggers a vicious cycle: the pool of valid critical points shrinks drastically at each subsequent layer, eventually stalling the extraction entirely.

Termination Condition and Naive Round Assessment. When the extraction reaches a certain depth where no new neurons can be clustered, a naive approach to termination would check whether all collected critical points have been assigned to specific neurons. Unassigned points would indicate that a neuron was missed somewhere in the network. However, the drawback of this naive assessment is severe: it only detects the missing neuron error at the very end of the extraction pipeline. Furthermore, it is entirely incapable of pinpointing the exact layer where the omission occurred. Consequently, the attacker is forced to collect a massive amount of additional critical points and exhaustively search for the missing neuron across all previous layers. This incurs unacceptable time and query costs, rendering polynomial-time extraction impossible.

Optimized Workflow via Early Error Detection. To avoid the catastrophic inefficiencies of the naive approach, an effective error detection method is strictly required to efficiently identify omissions in the immediately preceding layer. Prior to clustering the critical points for layer i, we additionally select several inputs that are not critical points of the previous layers (which can be easily filtered by computing the activation states of previously recovered neurons) and execute an *Error Detection* algorithm.

If missing neurons are detected in layer $i-1$, we collect additional critical points and re-execute the parameter recovery strictly for that specific layer, thereby halting error propagation immediately. Otherwise, we safely proceed to cluster the critical points belonging to the target layer i. In Section 3.3, we will prominently introduce our novel and efficient Error Detection algorithm specifically tailored for the logit-output setting.

Final Output Extraction. Once a layer L is successfully validated and no new set of critical points can be clustered for layer $L+1$, we extract the parameters of the final output layer via least squares, and the entire process successfully terminates. This adaptive workflow allows us to dynamically discover the network depth and completely extract all network parameters within polynomial time.

3.3 Error Detection in the Logit-Output Setting

Definition 7 (Space of Control and Degrees of Freedom). *Let F_x^{i-1} denote the linear transformation mapping the network's input to the input of layer i within the linear neighborhood of x. The* space of control *for layer i around input x, denoted by V_x^{i-1}, is defined as the range of the linear transformation F_x^{i-1}. The dimension of this space is called the number of* degrees of freedom *for layer i with input x, and is denoted by $\tilde{d}_x^{i-1} = rank(F_x^{i-1})$.*

The space of control represents the vector space containing all possible small changes (perturbations) at the input that can successfully propagate to layer i without altering the activation states of the preceding neurons. For a critical point x corresponding to a target neuron in layer i, the degree of freedom $\tilde{d}_x^{i-1}$ fundamentally determines the maximum number of weights that can be uniquely recovered during the differential attack.

Because the ReLU activation functions in layers 1 to $i-1$ force certain neurons into an inactive state (mapping their corresponding rows to the zero vector), the rank of the transformation matrices can never increase when multiplied together. Consequently, the number of degrees of freedom monotonically decreases or remains constant as i increases. In practice, $\tilde{d}_x^{i-1}$ is typically bounded by the initial input dimension d_0 and determined by the minimum number of active neurons per layer among all preceding layers 1 to $i-1$. Therefore, we have $\tilde{d}_x^{i-1} \leq \min(d_0, \min_{1 \leq k < i} d_{\text{active}}^k)$, though in cases of linear dependence, it can be strictly smaller.

Consider the recovery of layer i ($i \geq 2$). The core of the differential extraction in the logit-output setting [3] is to build and solve a system of linear equations for a weight vector A_1^i. By considering a set of basis vectors $H \subseteq \mathbb{R}^{d_{i-1}}$ and perturbing the intermediate input y along H and $-H$ with a small step size ϵ, we obtain the differentials α_j from the oracle. The recovery relies on constructing the following linear system:

$$\begin{cases} g_1^i \cdot A_1^i \cdot H_1 = \alpha_1 \\ \vdots \\ g_1^i \cdot A_1^i \cdot H_j = \alpha_j \end{cases} \tag{3.1}$$

By the solvability condition of the linear system, the differential attack requires $d(H_j) = d_{\text{active}}^{i-1} \leq \tilde{d}^{i-2}$ to successfully solve for the parameters A_1^i. Empirical validations [1] show that even in deep layers, the degrees of freedom stably maintain a sufficient proportion of the layer width, making this condition satisfiable for non-highly expansive architectures.

Now we mathematically analyze the scenario central to the limited architecture knowledge setting: **a neuron is missing in layer $i-1$.**

Let $\hat{y} \in \mathbb{R}^{d_{i-1}-1}$ denote the incompletely recovered input vector, and $y_{miss} \in \mathbb{R}$ denote the activation of the missing neuron. The true input vector propagating into layer i is exactly $y = [\hat{y}^T, y_{miss}]^T$. Correspondingly, the true weights of the target neuron in layer i can be partitioned as $A_1^i = [\hat{A}_1^i, A_{miss}^i]$.

When the attacker samples input perturbations to construct the basis matrix, they can only compute the dimension-deficient basis H' utilizing the incompletely recovered parameters of layer $i-1$. However, the black-box oracle $\mathcal{O}$ computes the output using the complete network. Therefore, the true differential vector α returned by the oracle implicitly contains the exact contribution of the missing neuron. Substituting the partitioned variables into Equation (3.1), the true algebraic relation generated by the oracle is:

$$\alpha = g_1^i \cdot (\hat{A}_1^i \cdot H' + A_{miss}^i \cdot H_{miss}) \tag{3.2}$$

where H_{miss} is the true, hidden perturbation induced on the missing neuron by the input queries.

In the extraction pipeline, the attacker is oblivious to $A_{miss}^i \cdot H_{miss}$ and attempts to construct the signature using the standard requirement: $d(H') = d(H' \mid \alpha)$. However, according to Equation (3.2), if the missing neuron is active at the critical point x (meaning $H_{miss} \neq \mathbf{0}$), the vector α strictly incorporates the vector H_{miss}.

Because H' is structurally oblivious to the missing dimension, α cannot be expressed as a linear combination of the row space of H'. This algebraic discrepancy inevitably leads to:

$$\text{rank}(H') < \text{rank}(H' \mid \alpha) \tag{3.3}$$

In this case, the system of equations inherently has no solution. We exploit this exact condition as an early-warning indicator. Because critical point clustering inherently relies on solving these signatures, the omission in layer $i-1$ is deterministically detected before the extraction of layer i proceeds, as formalized in Algorithm 1.

3.4 Noise Filtering and Redundant Neurons

Filtering noise components. We can incorporate the filtering procedures of [10] into critical point clustering to reduce interference from subsequent layers. These procedures first test candidate hyperplanes to identify deeper critical points, use subspace intersections to address rank deficiency, and then filter deeper components using their noise ratios, sizes, and unrecovered signature entries. The noise

Algorithm 1 Error Detection in the Logit-Output Setting

Require: A set of critical points $\mathcal{C}$ for layer i, recovered signature matrix $\tilde{A}^{i-1}$ for layer $i-1$, precise differentials α.
Ensure: **True** if extraction is consistent, **False** if a missing neuron is detected in layer $i-1$.
1: Let $d^{i-1}_{\text{active}} \leftarrow$ number of active neurons computed via $\tilde{A}^{i-1}$
2: Initialize basis matrix H' utilizing $\tilde{A}^{i-1}$ evaluated at $\mathcal{C}$
3: Compute the dimension of the basis: $d_{H'} \leftarrow \text{rank}(H')$
4: Compute the dimension of the augmented matrix: $d_{aug} \leftarrow \text{rank}(H' \mid \alpha)$
5: **if** $d_{H'} == d_{aug}$ **then return True** ▷ System is solvable, no missing neuron
6: **else return False** ▷ System has no solution, missing active neuron detected
7: **end if**

ratio measures merges with identified deeper critical points relative to component size. They provide an effective practical means of reducing noise, although their criteria do not guarantee that every retained component belongs to the target layer. Under limited architecture knowledge, filtering must therefore be combined with completeness checks rather than selecting a fixed number of components using the unknown target width.

Impact on Subsequent Signature Recovery Suppose all genuine neurons of layer i have been correctly recovered including r redundant neurons. Let $y \in \mathbb{R}^{d_i}$ and $z \in \mathbb{R}^r$ denote their respective activations. For a genuine neuron j in layer $i+1$, collect the corresponding perturbations into $H_y \in \mathbb{R}^{d_i \times m}$ and $H_z \in \mathbb{R}^{r \times m}$ for a consistent local differential system. Absorb the common downstream factor into the true signature $a = g^{i+1}_j A^{i+1}_j$. Since the true neuron takes only y as its input, the observed differential vector satisfies

$$\alpha = aH_y. \tag{3.4}$$

The recovery algorithm nevertheless includes all candidate coordinates and solves for $[u\ v]$, where v contains the weights associated with redundant neurons:

$$uH_y + vH_z = \alpha \quad \Longrightarrow \quad (u-a)H_y + vH_z = 0. \tag{3.5}$$

Therefore the solved weights necessarily satisfy $v = 0$. If H_y also has full row rank on the genuine coordinates being recovered, then $u = a$. From this, we can also infer that redundant neurons do not actually affect the recovery of subsequent neurons.

4 Experiments

To empirically validate the efficacy of our proposed framework, we conduct comprehensive parameter extraction experiments. Adhering to the standard evaluation pipeline established by Carlini et al. [2,8], we first train classification models

on the widely benchmarked CIFAR-10 dataset, and subsequently subject these trained models to our extraction attack. While we benchmark general extraction success using standard metrics consistent with prior literature, our evaluation specifically highlights and rigorously assesses the diagnostic accuracy of our novel Error Detection algorithm.

4.1 Experimental Setup and Target Architecture

Recognized as a fundamental benchmark in computer vision, CIFAR-10 provides a balanced subset of the 80 Million Tiny Images collection. The dataset consists of 60,000 RGB images (32×32 pixels) distributed across ten distinct object categories (such as airplanes, cats, and frogs), partitioned into 50,000 training samples and 10,000 testing samples. Flattening the spatial dimensions of each image yields an initial input vector of 3,072 features.

To ensure our evaluations are directly comparable to recent state-of-the-art studies [10], we deploy a fully-connected ReLU network of a similar scale. The target model architecture is defined as follows:

$$3072 - 512 \times 3 - 64 - 10$$

Specifically, the network processes the 3,072-dimensional input through four hidden layers—comprising three consecutive layers of 512 neurons each, followed by a narrower layer of 64 neurons—before culminating in a 10-neuron output layer equipped with a softmax activation to predict the CIFAR-10 classes. All hidden neurons utilize the ReLU activation function. In total, this configuration encompasses approximately 1.1 million parameters. And the trained target model achieves a baseline test accuracy of 0.52. This performance metric aligns well with the typical generalization capabilities of similarly scaled Multilayer Perceptrons (MLPs) reported in existing extraction literature [3, 10].

Additional model for the clustering experiment. To examine neuron omission and interference-induced neuron redundancy during parameter extraction in the limited architecture knowledge setting, we additionally evaluate a fully connected ReLU network with architecture

$$64 - 64 \times 4 - 10.$$

The model has a 64-dimensional input, four hidden layers of 64 neurons each, and a linear output layer producing ten output logits, with 17,290 trainable parameters including biases.

4.2 Neuron Omission and Redundancy under Limited Architecture Knowledge

This experiment examines whether neuron omission and neuron redundancy caused by interference from other neurons readily arise during neuron signature recovery in the limited architecture knowledge setting. Here, neuron redundancy refers to accepting spurious candidate neurons into the target layer.

Table 2: Clustering results for the $64 - 64 \times 4 - 10$ model.

Layer	Points	Components	Target neurons	Noise components	Coverage
1	2,000	64	64	0	100%
1	5,000	64	64	0	100%
1	10,000	64	64	0	100%
2	2,000	62	62	0	96.88%
2	5,000	64	63	1	98.44%
2	10,000	66	64	2	100%
3	2,000	56	56	0	87.50%
3	5,000	65	62	3	96.88%
3	10,000	72	63	9	98.44%
4	2,000	11	11	0	17.19%
4	5,000	31	30	1	46.88%
4	10,000	51	49	2	76.56%

For the $64 - 64 \times 4 - 10$ model, we cluster nested samples of 2,000, 5,000, and 10,000 critical points drawn from a pool of 100,000 points. For each target layer, the parameters of all preceding layers are supplied from the true model to isolate critical point clustering from errors in preceding-layer parameter recovery. Ground-truth parameters are used for evaluation to distinguish target-neuron components from noise components, which combine components belonging to other layers and components mixing points from multiple layers.

Table 2 shows that both failure modes arise in this experiment. With 2,000 critical points, layers 2–4 omit 2, 8, and 53 target neurons, respectively. Increasing the sample size improves coverage, but layers 3 and 4 still omit 1 and 15 neurons at 10,000 points. Meanwhile, noise components appear in all three layers as the sample size increases. If accepted as target neurons, these components introduce redundancy into the inferred layer.

Omission and redundancy can also coexist and conceal one another. At layer 2 with 5,000 points, the 64 components comprise 63 target neurons and one noise component. At layer 3 with 10,000 points, 72 components represent only 63 target neurons, with nine noise components. These results indicate that in the limited architecture knowledge setting, neuron omission and neuron redundancy are relatively commoni. Therefore, it is highly necessary to introduce an error detection algorithm to detect the neuron recovery situation.

4.3 Evaluation Metrics for Error Detection

To rigorously evaluate the performance, sensitivity, and efficiency of our proposed error detection mechanism, we define three core evaluation metrics. Let $\mathcal{T}$ denote a set of independent extraction trials for layer i. For each trial $t \in \mathcal{T}$, let $Y_t \in \{0, 1\}$ be the ground-truth indicator, where $Y_t = 1$ denotes that there are actual missing neurons in the preceding layer $i - 1$. Let $\mathcal{C}_t$ be the set of critical points

queried during trial t. Our detection algorithm applies an evaluation function $\mathcal{E} : \mathcal{C}_t \to \{0, 1\}$, where $\mathcal{E}(c) = 1$ indicates that a dimensional anomaly (error) is detected at point c. The overall trial-level prediction is denoted as $\hat{Y}_t \in \{0, 1\}$, where $\hat{Y}_t = 1$ if $\exists c \in \mathcal{C}_t$ such that $\mathcal{E}(c) = 1$.

Definition 8 (Error Detection Recall, EDR). *The Error Detection Recall evaluates the sensitivity of the algorithm. It is defined as the ratio of correctly identified erroneous layers to the total number of layers that actually contain missing neurons:*

$$EDR = \frac{\sum_{t \in \mathcal{T}} \mathbb{I}(Y_t = 1 \wedge \hat{Y}_t = 1)}{\sum_{t \in \mathcal{T}} \mathbb{I}(Y_t = 1)}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

EDR is the most critical metric for evaluating the extraction fidelity. It answers the fundamental question: *When an omission actually occurs, how effectively can our algorithm catch it?* A high EDR guarantees that the extraction pipeline will almost never propagate corrupted parameters to deeper layers.

Definition 9 (Error Detection Specificity, EDS). *The Error Detection Specificity measures the algorithm's ability to correctly identify consistent (error-free) layers without issuing false alarms. It is defined as the ratio of true negative detections to all actual error-free extraction trials:*

$$EDS = \frac{\sum_{t \in \mathcal{T}} \mathbb{I}(Y_t = 0 \wedge \hat{Y}_t = 0)}{\sum_{t \in \mathcal{T}} \mathbb{I}(Y_t = 0)}$$

EDS strictly evaluates the robustness of our framework against false positives. A false positive ($\hat{Y}_t = 1$ when $Y_t = 0$) occurs when the algorithm erroneously estimates a structural omission, forcing the pipeline to waste significant computational resources and query budgets on unnecessary backtracking. Maintaining a high EDS demonstrates that our detection mechanism can effectively distinguish between inherent depth-induced rank deficiency and genuine structural omissions.

Table 3: Layer-wise weight extraction results on CIFAR-10 networks under different output settings.

Metric	L1	L2	L3	L4
Error Detection Recall	100%	100%	100%	100%
Error Detection Specificity	1.91%	1.94%	2%	2.03%

4.4 Results

Table 3 summarizes the layer-wise performance of our proposed error detection algorithm applied to the CIFAR-10 trained neural network. Because our adaptive workflow utilizes the same underlying signature and sign recovery algorithms as established cryptanalytic frameworks [1, 3], the core parameter extraction mechanics remain fundamentally consistent. Consequently, we do not separately report the overall time and query cost metrics; the model parameter extraction inherently maintains both polynomial time and polynomial query complexity.

The core metrics of interest in this paper—Error Detection Recall and Error Detection Specificity—demonstrate the robust diagnostic capabilities of our approach. Most notably, the algorithm achieves a 100% Error Detection Recall across all hidden layers. This perfectly confirms the efficacy of our proposed mechanism, guaranteeing that no silent neuron omissions can bypass the detection phase to corrupt downstream parameter recovery. Additionally, the Error Detection Specificity is recorded consistently between 1.91% and 2.03% as the extraction progresses through the network, validating the algorithm's stability across different depths.

5 Conclusions

This work advances the field of cryptanalytic extraction of neural networks. Focusing on fully connected ReLU-based architectures, we introduce the novel direction of limited architecture knowledge and investigate its impact on prior model parameter recovery attacks. We further propose a novel error detection algorithm for neuron omission to enhance existing parameter recovery attacks. The effectiveness of our approach is demonstrated through experiments on ReLU-based neural networks trained on CIFAR-10. To the best of our knowledge, this is the first work to provide empirical evidence that a functionally equivalent network can be successfully recovered under the limited architecture knowledge scenario. Specifically, our experimental results demonstrate that for a four-hidden-layer network comprising over 1.1 million parameters, our framework achieves a 100% Error Detection Recall, successfully identifying all structural omissions on-the-fly while maintaining polynomial time and query complexity.

However, several open problems remain in this paper for future research. First, extending our error detection mechanism to the more restrictive hard-label setting—where the absence of exact output logits makes it highly difficult to algebraically verify dimensional consistency—remains a challenging and important direction. Finally, the distribution of critical points is not uniform across each hidden layer. Previous works often set the number of acquired critical points as $\mathcal{O}(N \log N)$ [3], based on the assumption that at least two critical points are assigned to each neuron, which does not align with practical scenarios. This discrepancy can lead to an increase in neuron absence errors. Therefore, establishing a more reasonable number of critical points to acquire or developing better critical point search methods could reduce redundant parameter extraction efforts and lower time costs.

A.1 Additional Definitions and Notation

Here we present some basic definitions and notations used throughout the manuscript which closely follows the terminology first presented in [3] and then adopted in paper [1,2,4].

Definition 10 (Fully Connected Layer). *The i-th* fully connected layer *of a neural network is a function $f_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ given by the affine transformation*

$$f_i(x) = A^{(i)}x + b^{(i)},$$

where $A^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and $b^{(i)} \in \mathbb{R}^{d_i}$ are, respectively, the weight matrix *and the* bias vector *of the i-th* layer*, and d_{i-1}, d_i are positive integers.*

Definition 11 (Neuron). *A* neuron *is a function determined by the corresponding weight matrix followed by an activation function. Particularly, the j-th neuron of layer i is the function η given by*

$$\eta(x) = \sigma(A_j^{(i)}x + b_j^{(i)}),$$

where $A_j^{(i)}$ and $b_j^{(i)}$ denote, respectively, the j-th row of $A^{(i)}$ and the j-th coordinate of $b^{(i)}$. An r-deep neural network has $N = \sum_{k=1}^{r} d_k$ neurons.

Definition 12 (Network Architecture). *The* architecture *of a fully connected neural network is described by specifying its number of layers along with the dimension d_i (i.e., number of neurons) of each layer $i = 1, \cdots, r+1$. We say that d_0 is the dimension of the inputs to the neural network and d_{r+1} is the number of outputs of the neural network.*

Definition 13 (Network Parameters). *The* parameters *θ of an r-deep neural network f_θ are the concrete assignments to the weights $A^{(i)}$ and biases $b^{(i)}$, $i \in \{1, 2, \ldots, r+1\}$.*

When working under the hard-label setting, the raw output $f(x)$ is processed before returning the result. We consider the processing employed in [4], which is presented in the definition below.

References

1. Canales-Martinez, I.A., Chavez-Saab, J., Hambitzer, A., Rodriguez-Henriquez, F., Satpute, N., Shamir, A.: Polynomial time cryptanalytic extraction of neural network models. Cryptology ePrint Archive (2023)
2. Carlini, N., Chavez-Saab, J., Hambitzer, A., Rodriguez-Henriquez, F., Shamir, A.: Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. In: Advances in Cryptology - EUROCRYPT 2025. Lecture Notes in Computer Science, vol. 15601, pp. 364–396. Springer (2025). https://doi.org/10.1007/978-3-031-91107-1_13

3. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Annual International Cryptology Conference. pp. 189–218. Springer (2020)
4. Chen, Y., Dong, X., Guo, J., Shen, Y., Wang, A., Wang, X.: Hard-label cryptanalytic extraction of neural network models. Cryptology ePrint Archive, Paper 2024/1403 (2024), https://eprint.iacr.org/2024/1403
5. Daniely, A., Granot, E.: An exact poly-time membership-queries algorithm for extraction a three-layer relu network. arXiv preprint arXiv:2105.09673 (2021), https://arxiv.org/abs/2105.09673
6. Foerster, H., Mullins, R.D., Shumailov, I., Hayes, J.: Beyond slow signs in high-fidelity model extraction. arXiv preprint arXiv:2406.10011 (2024). https://doi.org/10.48550/ARXIV.2406.10011, https://doi.org/10.48550/arXiv.2406.10011
7. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 1345–1362 (2020)
8. Lin, Z., Memisevic, R., Konda, K.: How far can we go without convolution: Improving fully-connected networks. arXiv preprint arXiv:1511.02580 (2015)
9. Liu, H., Siproudhis, A., Boura, C., Peyrin, T.: Model extraction of convolutional neural networks with max-pooling. Cryptology ePrint Archive, Paper 2026/464 (2026), https://eprint.iacr.org/2026/464
10. Liu, H., Siproudhis, A., Experton, S., Lorenz, P., Boura, C., Peyrin, T.: Navigating the deep: End-to-end extraction on deep neural networks. In: Advances in Cryptology - EUROCRYPT 2026 (2026), https://eprint.iacr.org/2026/296
11. 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. OpenReview.net (2024)
12. Milli, S., Schmidt, L., Dragan, A.D., Hardt, M.: Model reconstruction from model explanations. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. pp. 1–9. ACM (2019)
13. Reith, R.N., Schneider, T., Tkachenko, O.: Efficiently stealing your machine learning models. In: Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society. pp. 198–210. ACM (2019)
14. Rolnick, D., Kording, K.P.: Reverse-engineering deep relu networks. In: International Conference on Machine Learning. pp. 8178–8187. PMLR (2020)
15. 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
16. Tramer, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 601–618 (2016)