Cryptanalytic Extraction of Convolutional Neural Networks

Xiaohan Sun[1,2,3], Hao Lei[1,2,3], Longxiang Wei[1,2,3], Xiaokang Qi[1,2,3], Kai Hu[1,3,4,5], Meiqin Wang[1,2,3], and Wei Wang[1,2,3,4(✉)]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China
xhansun@mail.sdu.edu.cn,leihao@mail.sdu.edu.cn,longxiangwei@mail.sdu.edu.cn, xiaokangqi@mail.sdu.edu.cn,kai.hu@sdu.edu.cn,mqwang@sdu.edu.cn, weiwangsdu@sdu.edu.cn
[2] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, 266237, China
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China.
[4] Quan Cheng Shandong Laboratory, Jinan, China
[5] Suzhou Research Institute, Shandong University, Suzhou, 215123, China

Abstract. Neural network model extraction attacks pose a serious threat to the intellectual property of deep learning models. While most prior work focuses on Fully Connected Networks (FCNs), effective extraction of Convolutional Neural Networks (CNNs) remains underexplored, particularly in the hard-label setting. In this work, we propose the first systematic method for the recovery of complete CNN parameters in such conditions. By reformulating convolutional layers as sparse Block Toeplitz with Toeplitz Blocks (BTTB) matrices, we extend the model extraction attack method from FCNs to CNNs. The proposed method supports both one- and two-dimensional CNNs, handling scenarios with multiple kernels, multi-channel structures, and average pooling. To enhance computational efficiency and scalability, a kernel-centric clustering algorithm is proposed to exploit kernel parameter sharing, and a Singular Value Decomposition (SVD)-based acceleration strategy is adopted to address the computational cost of large sample sets. Moreover, we perform experiments to demonstrate that our method accurately and efficiently extracts CNN parameters, including multi-channel, multi-kernel and average-pooling layers, with a worst-case relative error of $2^{-17.75}$ and up to $2^{9.26}$ speedup, and recover large models LeNet-5 within practical runtime.

Keywords: Neural Network Extraction · ReLU-based Convolutional Neural Networks · Hard-label Attack

1 Introduction

The problem of neural network model extraction, which seeks to recover a model's trained parameters using oracle query access. This field originated in

the early 1990s at the intersection of cryptography, computational learning theory, and neural computation [9,2]. Foundational studies by Fefferman [9] and by Blum and Rivest [2] investigated the theoretical identifiability and computational hardness of recovering neural architectures, establishing an early link between model extraction and cryptographic key recovery. Concurrently, Baum proposed the first polynomial-time learning algorithms for this task under the Probably Approximately Correct (PAC) framework [1].

Since 2016, numerous studies [7,11,14] have revisited the problem from a black-box perspective, focusing on recovering model parameters, achieving sufficient numerical precision. Contemporary attacks, for instance, operating in the logit-output setting, can precisely recover network parameters in polynomial time, yielding models that are parameter-identical to the victim [15]. While these approaches demonstrate high fidelity, their success hinges on access to continuous confidence scores, a condition rarely met in practical, hard-label scenarios.

To overcome this limitation, Chen *et al.* [5] pioneered the analysis of decision boundary points in the hard-label setting. However, their attack could deal only with tiny, binary-class networks because of the exponential execution time and the lack of an efficient feature-recovery mechanism for individual neurons. Consequently, their method was only applicable to small, binary-class networks. Later, Carlini *et al.* [3] developed a framework that achieves polynomial-time recovery of all hidden parameters, and deeper networks.

A series of recent works have successfully extracted the parameters of Fully Connected Networks (FCNs) using cryptanalytic methods in hard-label settings. However, for other not fully connected Networks, such as Convolutional Neural Networks (CNNs), their extraction mainly remains unexplored. With the notable exception of a recent side-channel attack [6], virtually no work has addressed the extraction of CNNs, which dominate modern AI applications such as computer vision, speech recognition, and signal processing. This research gap motivates our work, which seeks to extend the theoretical and algorithmic foundations of model extraction from FCNs to CNNs by analyzing their intrinsic convolutional structure.

1.1 Our Contributions

The parameter-sharing architecture of CNNs fundamentally conflicts with the neuron independence assumption underlying the hard-label attack proposed by Carlini *et al.* [3], so that their attack cannot be directly applied to CNNs. To overcome this limitation, we reformulate convolutional layers as sparse matrices with Block Toeplitz with Toeplitz Blocks (BTTB) structure, transfering CNNs to the same mathematical framework as FCNs. However, with the expansion of the network, the recovery time of the original algorithm becomes excessively long. To address scalability, we introduce two complementary acceleration techniques, kernel centric dual point clustering and Singular Value Decomposition (SVD)-based covariance reduction, which effectively shrink the search space and substantially reduce memory overhead. In this way, the first practical parameter extraction model of hard-label CNNs is obtained, achieving up to $2^{9.26}$ times

faster processing and enabling full recovery of the LeNet-5 model within a practical runtime. Moreover, the recovered parameters achieve a worst-case relative error of $2^{-17.75}$ compared to the original ones, demonstrating both high numerical precision and scalability of our architectures. The contribution of this work includes the following.

1. **Convolutional Kernel Recovery via BTTB Matrix Formulation.** In order to bridges the gap between convolutional and fully connected layers, we leverage the intrinsic parameter-sharing property of CNNs, and adpot the BTTB structure to formalize convolutional operations as sparse matrices, allowing convolutional layers to be treated as specialized fully connected layers. Consequently, the parameter extraction problem is reframed and transformed from the recovery of numerous independent neuron weights to the reconstruction of a smaller set of core convolutional kernel parameters. Moreover, the clustering and signature recovery procedures are adjusted accordingly.
2. **Kernel-Centric Dual-Point Clustering Algorithm.** To overcome the inefficiency of traditional neuron-wise matching, we propose a kernel-centric dual-point clustering algorithm. Instead of recovering every individual neuron, our method identifies a single representative cluster of points for each convolutional kernel, which drastically reduces the search space, experimentally decreasing the number of clusters from 530 to 1 in some networks, and significantly improves computational efficiency. Subsequently, the complete weight matrix can be deterministically reconstructed from these representative neurons.
3. **SVD-Based Acceleration Technique.** We apply SVD to the covariance matrix of dual points to reduce both memory usage and computational overhead. This acceleration is mathematically equivalent to the original formulation and offers significant efficiency improvements, especially in high-sample regimes. Combined with kernel-centric clustering, SVD-based acceleration substantially speeds up the entire signature recovery process.

To evaluate the applicability and scalability of our model across various network architectures, we test models with different numbers of layers and convolutional kernel configurations, covering the recovery of parameters from different layers. All experiments are conducted on a machine equipped with an Intel i7 processor, 16 GB of RAM, and an NVIDIA RTX 3060 GPU. Table 1 summarizes the runtime comparison between our method and that of Carlini *et al.* [3]. In the **Architecture** column, the first entry denotes the original CNN architecture, with the corresponding equivalent FCN architecture shown in parentheses. As shown in Table 1, our method successfully extracts the parameters of four representative CNN models and substantially improve computational efficiency, which consistently outperforms the attack of Carlini *et al.* [3] across all evaluated architectures.

For single-layer parameter recovery, the speedup on small networks is approximately $2^{3.8}$, while for medium-scale architectures, it reaches up to $2^{9.26}$. The software implementation is available at:

https://anonymous.4open.science/r/cnn-extraction-93C4

Table 1. Runtime of Single-Layer Signature Recovery on CNNs.

Architecture (CNN–FCN)	**Approach**	$t_{\mathbf{dual}}$ (s)	$t_{\mathbf{cluster}}$ (s)	$t_{\mathbf{unify}}$ (s)	m	N	$t_{\mathbf{sig}}$ (s)
80–49–18–10 (**240–49**–18–10)	[3]	$2^{9.91}$	$2^{13.88}$	$2^{5.12}$	49	–	$2^{13.98}$
	This paper	$2^{9.91}$	$2^{7.39}$	$2^{-0.03}$	1	30	$2^{10.14}$
80–49–18–10 (**240–147**–18–10)	[3]	$2^{10.27}$	$2^{15.16}$	$2^{10.79}$	147	–	$2^{15.28}$
	This paper	$2^{10.27}$	$2^{10.58}$	$2^{2.05}$	16	30	$2^{11.43}$
784–**657–530**–403–10 (784–**657–530**–403–10)	[3]	$2^{12.38}$	$2^{23.34}*$	$2^{13.67}*$	530	–	$2^{23.47}*$
	This paper	$2^{12.38}$	$2^{13.74}$	$2^{2.55}$	1	30	$2^{14.21}$
LeNet-5 (784–4704-**1176-1600** -400-120–84–10)	[3]	$2^{14.58}$	$2^{26.04}*$	$2^{15.36}*$	1600	–	$2^{26.04}*$
	This paper	$2^{14.58}$	$2^{19.34}$	$2^{8.49}$	150	7	$2^{19.39}$

Bold layers in *Architecture* indicate the single layer being recovered.
−: the corresponding parameter is not considered.
m: the target number of clusters; N: the clustering threshold.
$*$: the theoretical runtime estimates. Due to excessive computation time, these values are theoretically estimated after partial calculations.

1.2 Organization

The remainder of this paper is organized as follows. Section 2 outlines the background of CNNs and provides a brief introduction of related works. Section 3 details our extraction framework, focusing on CNN Signature Recovery. The signature recovery process primarily comprises four key components: Convolutional Kernel Recovery via BTTB Matrix Formulation, Kernel-Centric Dual-Point Clustering, SVD-Based Acceleration, and Average Pooling Extraction. Experimental evaluations are presented in Section 4, and the paper is concluded in Section 5.

2 Preliminaries

2.1 Convolutional Neural Networks

FCNs are characterized by a dense connection pattern, where each neuron in a layer connects to every neuron in the subsequent layer. While this structure

allows for powerful global feature integration, it disregards local data correlations and leads to an exponential growth in parameters when handling high-dimensional inputs. Consequently, FCNs often suffer from computationally expensive training and poor scalability, limiting their application in many modern tasks.

CNNs were developed to overcome these limitations by employing three core mechanisms: local connectivity, parameter sharing, and pooling. These principles enable the efficient extraction of structured features and substantially reduce the number of trainable parameters while preserving essential information. Unlike FCNs, CNNs are inherently designed to process local data structures, and they are typically categorized by the dimensionality of their input. For instance, 2D CNNs [16] are designed for spatial data like images, capturing hierarchical visual features such as edges and textures. In contrast, 1D CNNs [17,10] process sequential data and are widely applied in time-series analysis, biomedical signal processing [13], and lightweight natural language tasks [8]

In the following, we use 1D CNNs as a representative case to briefly describe their architectural structure.

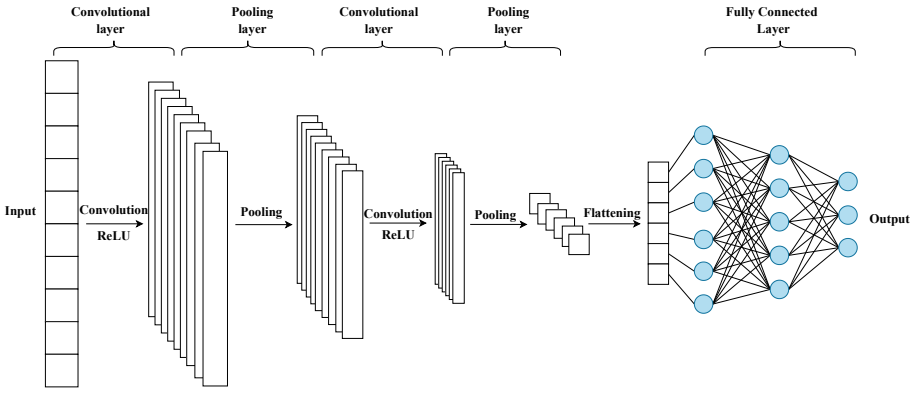

Fig. 1. Overall structure of a 1D CNN.

Overall Architecture A typical 1D CNN consists of Convolutional, Pooling, and Fully Connected Layers (Fig. 1). The general workflow is:

1. **Input Layer:** Receives raw 1D sequences (e.g., time-series or embeddings).
2. **Convolutional Layer:** Applies local convolutions with shared kernels and nonlinear activation.
3. **Pooling Layer:** Downsamples to reduce computation and improve invariance.
4. **Flattening:** Converts multi-channel maps into a single vector.
5. **Fully Connected Layer:** Aggregates features and outputs class scores.

Convolutional Layer The convolutional layer is the computational core of a CNN, responsible for extracting structured features from input data. Given an input sequence

$$X \in \mathbb{R}^{C_{\text{in}} \times L_{\text{in}}}, \tag{1}$$

with input channels C_{in} and length L_{in}, the convolutional layer produces an output

$$Y \in \mathbb{R}^{C_{\text{out}} \times L_{\text{out}}}, \tag{2}$$

where C_{out} denotes the number of output channels and L_{out} the resulting sequence length.

This operation is defined by two foundational principles: *local connectivity*, where each output neuron is connected to only a small region of the input (its receptive field), and *parameter sharing*, where the same kernel is applied across all positions in the sequence. The behavior of a 1D convolutional layer is determined by several key hyperparameters, summarized below.

Kernel and Channels. The kernel size (K) specifies the length of the filter, which defines the receptive field of each neuron. The number of input channels (C_{in}) must match the depth of the input tensor, while the output channels (C_{out}) correspond to the number of distinct filters applied to the input. Each filter is trained to detect a specific pattern, producing a corresponding feature map.

Definition 1. *Formally, for each output channel j, a unique filter, consisting of a weight tensor $A_j \in \mathbb{R}^{C_{in} \times K}$ and a bias term b_j, is convolved with the input X to produce an output feature map Y_j:*

$$Y_j = A_j * X + b_j, \quad j = 1, 2, \ldots, C_{out}. \tag{3}$$

Stride and Padding. The stride (S) dictates the step size at which the kernel slides across the input, and padding (P) refers to the number of zeros appended to the sequence boundaries. These hyperparameters jointly determine the output length L_{out} via the formula:

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} - K + 2P}{S} \right\rfloor + 1. \tag{4}$$

Common strategies include valid padding ($P = 0$), which reduces the output dimension, and *same* padding, which sets P to preserve the input length (e.g., $P = (K-1)/2$ for $S = 1$).

Spatial Arrangement. The interplay of kernel size, stride, and padding determines the degree to which the receptive fields of adjacent output neurons overlap. This overlap is crucial, as it ensures smooth feature transitions and enables the network to robustly capture patterns that are continuous or correlated across local regions.

Illustrative Example of 1D Convolution Fig. 2 provides a concrete illustration of a one-dimensional, two-channel convolutional operation. The example layer is configured with an input of $C_{\text{in}} = 2$ channels and length $L_{\text{in}} = 9$, and it produces an output of $C_{\text{out}} = 2$ channels and length $L_{\text{out}} = 4$, using a kernel of size $K = 3$, a stride of $S = 2$, and no padding ($P = 0$). As the figure depicts, each output filter K_j is itself a collection of channel-specific kernels ($k_{j,c,:}$), one for each input channel c. These kernels operate in concert, aggregating spatially local information across all input channels to produce a single output feature map. By stacking such convolutional layers, the network is able to construct hierarchical feature representations of progressively higher abstraction.

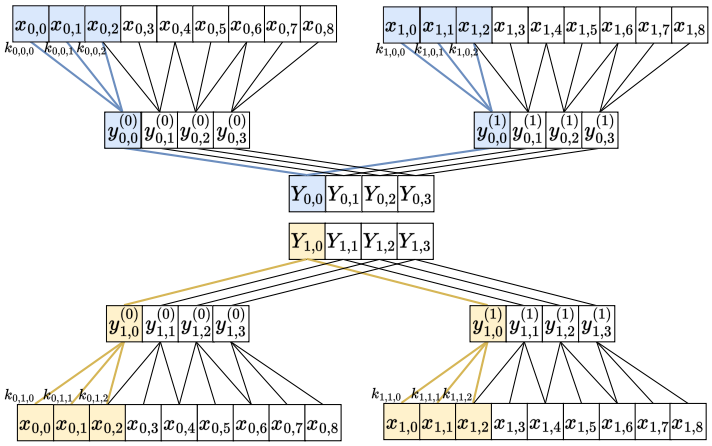

Fig. 2. The convolutional layer sample.

2.2 Extraction of FCNs in Hard-Label Setting

The work of Carlini *et al.* [3] shows that the parameters of a deep neural network can be recovered by analyzing the geometry of its decision boundary in the hard-label setting. Their extraction attack is the first cryptanalytic method capable of recovering modern deep neural networks using only a polynomial number of queries and polynomial time. The framework conducts a layer-by-layer extraction procedure consisting of two main phases: Signature Recovery and Sign Recovery. Signature recovery identifies each neuron's parameters up to an unknown sign, corresponding to the recovery of its critical hyperplane. Sign recovery establishes the correct orientation of the hyperplane, an essential step for accurately reproducing the neuron's activation behavior under the ReLU nonlinearity.

As our model propose a distinct sign recovery strategy, the signature recovery phase in Carlini et al.'s layer-by-layer extraction procedure is briefly recalled, and for more detail of their sign recovery phase please refer to [3].

Signature Recovery The objective of the Signature Recovery phase is to recover the parameters of each neuron in the target layer. Specifically, this process recovers the neuron's critical hyperplane, which is defined by its weights and bias, albeit only up to an unknown scaling factor.

In the hard-label setting, the attacker can only query an oracle $\mathcal{O}$ for its final classification label, without access to the numeric values of the outputs [4]. This fundamental restriction renders model extraction methods, which rely on gradients or precise numerical output variations, ineffective.

To address this challenge, the work by Carlini *et al.* [3] exploits the concept of dual points, which are derived from the piecewise linear structure of ReLU networks. This method is founded on the principle that the critical hyperplanes of all neurons partition the input space into convex cells. Within any single region, the decision boundary is strictly linear. Consequently, a change in the decision boundary's orientation, or a bend, can occur only when the input crosses between these regions, which signifies an intersection with a neuron's critical hyperplane. The location of this observable bend is defined as a dual point, representing the precise intersection of the decision boundary and an otherwise unobservable critical hyperplane. By systematically locating these dual points, an attacker can therefore indirectly infer the positions of the neurons' critical hyperplanes.

Figure 3 illustrates this geometric relationship in three dimensions. The vertical yellow plane represents the unknown critical subspace of a neuron, while the two nearly horizontal sheets are adjacent patches of the decision boundary. These patches exhibit different local orientations because they lie on opposite sides of the critical plane, where the neuron's activation state switches. While their intersection forms a one-dimensional line of dual points contained within the critical plane, the plane's rotational orientation around this line remains uncertain. This ambiguity is resolved, and the neuron's critical hyperplane fully determined, by finding a another instance of criticality for the same neuron elsewhere in the input space.

The procedure for finding a dual point begins with a binary search between two differently-labeled inputs to precisely locate a point on the decision boundary. The attacker then crosses this boundary until its path deviates.This bend marks the location of the dual point.

Once a large collection of dual points is gathered, the attack proceeds to the clustering and parameter recovery phase. First, points are clustered by neuron using the IsConsistent algorithm 2(show in Appendix). This algorithm determines if two dual points belong to the same neuron by computing the rank of the vector space spanned by their respective dual spaces, where each dual space is the intersection of local decision boundary patches. After clustering, the complete critical hyperplane is reconstructed by merging the dual spaces of all points from a single cluster. The normal vector of the resulting hyperplane corresponds to the weight of the neuron, thus completing the recovery of its signature.

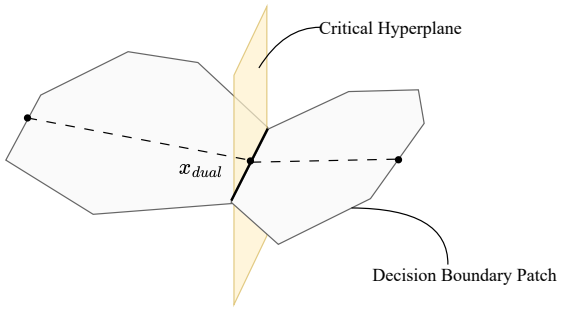

Fig. 3. A neuron's critical hyperplane intersects with two adjacent patches of the decision boundary

2.3 Adversarial Goals and Assumptions

According to our security model, the attacker can adaptively select queries x as inputs to an oracle $\mathcal{O}$, which returns the label of x produced by the target CNN f_θ. The attacker's objective is to recover a set of parameters $\hat{\theta}$, that is functionally equivalent[6] to the original parameters θ.

We assume the attacker has the following capabilities:

- **Knowledge of the architecture.** The attacker knows the full network architecture, including layer types (convolutional, average pooling, or fully connected), the number of neurons and channels per layer, and key hyperparameters such as kernel size (K), stride (S), and padding (P) for all convolutional and pooling layers.
- **Full-domain inputs.** The attacker can query arbitrary inputs from the full input domain $\mathbb{R}^{d_0}$.
- **Precise computations.** The CNN is specified and evaluated using sufficiently high-precision floating-point arithmetic.
- **Activation functions.** ReLU activation functions are applied after convolutional layers but not after average-pooling layers. Our formulation accommodates any number of classes.

3 Extraction attack on Convolutional Neural Networks

The approach proposed by Carlini *et al.* [3], based on decision boundary geometry, enables complete recovery of all neuron parameters using only hard-label outputs. The success of such methods relies on the *global linear mapping* property of FCNs, where each neuron corresponds directly to an independent critical

[6] *Functional equivalence* means that although convolution and its BTTB-based linear reformulation differ in computational form, they implement the same input–output mapping and therefore produce identical outputs for any given input.

hyperplane. However, CNNs used in computer vision differ fundamentally. Convolutional layers obey *local connectivity* and *parameter sharing*: each neuron depends only on a local receptive field, and identical kernels are applied across spatial positions. This structure breaks the FCN assumption of one neuron per hyperplane, posing a challenge to extending extraction algorithms that rely on the geometric analysis of decision boundaries.

3.1 Attack Overview

This chapter presents an attack framework for extracting CNN parameters in the hard-label setting. The recovery proceeds layer-by-layer and consists of two main stages: Signature Recovery and Sign Recovery. To resolve the conflict between CNN parameter sharing and the neuron independence assumption in [3], the Signature Recovery stage reformulates each convolutional layer as an equivalent linear mapping represented by a structured sparse BTTB matrix. Based on this linearized model, dual points are obtained by identifying a bend on the decision boundary. A kernel-centric clustering strategy then groups these dual points by their corresponding convolutional kernel, and the critical hyperplane of each kernel is reconstructed by merging the subspaces spanned by its cluster. An SVD-based acceleration further improves stability and efficiency. For average-pooling layers, their linearity and known architecture enable analytical reconstruction without boundary analysis. Finally, the Sign Recovery stage completes the process via an exhaustive search to determine each kernel's sign.

3.2 Linear Reformulation of Convolutional Kernel

To resolve the structural incompatibility between parameter sharing in CNNs and the assumption of neuron independence in attacks on FCNs [3], we reformulate each convolutional layer as a functionally equivalent linear mapping. We show that any convolution can be expressed as a structured sparse linear transformation, represented by a *Block-Toeplitz with Toeplitz Blocks* (BTTB) matrix. This representation establishes a deterministic mapping from convolutional kernels to their equivalent fully connected weight matrices.

General Form Let the input $\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times L_{\text{in}}}$, the kernel tensor $\mathbf{K} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K}$, and the output $\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times L_{\text{out}}}$, where the scalar K denotes the kernel size. For a convolution with stride 1 and no padding, the output length L_{out} is given by:

$$L_{\text{out}} = L_{\text{in}} - K + 1. \tag{5}$$

This matrix acts on the channel-wise vectorized input $\text{vec}(\mathbf{X})$, where each block $\mathbf{T}(\mathbf{K}_{i,j})$ acts only on the corresponding input channel vector $\mathbf{x}_j$.

$$\text{vec}(\mathbf{X}) = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{C_{\text{in}}-1} \end{bmatrix}, \quad \mathbf{x}_j \in \mathbb{R}^{L_{\text{in}}}. \tag{6}$$

The transformation can then be expressed in vectorized form as:

$$\mathrm{vec}(\mathbf{Y}) = \mathbf{T}_{\mathrm{BTTB}}(\mathbf{K})\,\mathrm{vec}(\mathbf{X}), \tag{7}$$

where $\mathrm{vec}(\cdot)$ denotes channel-wise vectorization. The matrix $\mathbf{T}_{\mathrm{BTTB}}(\mathbf{K})$ is composed of $C_{\mathrm{out}} \times C_{\mathrm{in}}$ Toeplitz submatrices, each encoding the local convolution between one input channel and one kernel slice.

Toeplitz Block Structure The convolution between a single input channel j and output channel i is performed by the corresponding kernel slice $\mathbf{K}i, j$. This operation is captured by the Toeplitz submatrix $\mathbf{T}(\mathbf{K}i, j) \in \mathbb{R}^{L_{\mathrm{out}} \times L_{\mathrm{in}}}$, which is constructed as follows:

$$\mathbf{T}(\mathbf{K}_{i,j}) = \begin{bmatrix} k_{i,j,0} & k_{i,j,1} & \cdots & k_{i,j,K-1} & 0 & \cdots & 0 \\ 0 & k_{i,j,0} & \cdots & k_{i,j,K-2} & k_{i,j,K-1} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & k_{i,j,0} & \cdots & k_{i,j,K-2} & k_{i,j,K-1} \end{bmatrix}. \tag{8}$$

Each row of this matrix corresponds to a spatially shifted receptive field, thereby encoding the local connectivity and parameter sharing intrinsic to convolution.

Block-Toeplitz with Toeplitz Blocks The complete BTTB matrix $\mathbf{T}_{\mathrm{BTTB}}(\mathbf{K})$, is a block matrix composed of the Toeplitz submatrices for every input-output channel pair:

$$\mathbf{T}_{\mathrm{BTTB}}(\mathbf{K}) = \begin{bmatrix} \mathbf{T}(\mathbf{K}_{0,0}) & \mathbf{T}(\mathbf{K}_{0,1}) & \cdots & \mathbf{T}(\mathbf{K}_{0,C_{\mathrm{in}}-1}) \\ \mathbf{T}(\mathbf{K}_{1,0}) & \mathbf{T}(\mathbf{K}_{1,1}) & \cdots & \mathbf{T}(\mathbf{K}_{1,C_{\mathrm{in}}-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{T}(\mathbf{K}_{C_{\mathrm{out}}-1,0}) & \mathbf{T}(\mathbf{K}_{C_{\mathrm{out}}-1,1}) & \cdots & \mathbf{T}(\mathbf{K}_{C_{\mathrm{out}}-1,C_{\mathrm{in}}-1}) \end{bmatrix}. \tag{9}$$

Example Consider a one-dimensional convolution with input length $L_{\mathrm{in}} = 10$, kernel size $K = 3$, input channels $C_{\mathrm{in}} = 3$, and output channels $C_{\mathrm{out}} = 3$. With stride $S = 1$ and no padding, the output length is $L_{\mathrm{out}} = L_{\mathrm{in}} - K + 1 = 8$. The input $\mathbf{X} \in \mathbb{R}^{3\times 10}$ and output $\mathbf{Y} \in \mathbb{R}^{3\times 8}$ are vectorized channel-wise as $\mathrm{vec}(\mathbf{X}) \in \mathbb{R}^{30}$,$\mathrm{vec}(\mathbf{Y}) \in \mathbb{R}^{24}$.

1. Toeplitz block construction. For each input–output channel pair (i, j), the convolution induced by the kernel slice $\mathbf{K}_{i,j} = (k_{i,j,0}, k_{i,j,1}, k_{i,j,2})$ can be written as a Toeplitz matrix $\mathbf{T}(\mathbf{K}_{i,j}) \in \mathbb{R}^{8\times 10}$.

For the present example with $L_{\mathrm{in}} = 10$ and kernel size $K = 3$, this matrix takes the explicit form

$$\mathbf{T}(\mathbf{K}_{i,j}) = \begin{bmatrix} k_{i,j,0} & k_{i,j,1} & k_{i,j,2} & 0 & \cdots & 0 \\ 0 & k_{i,j,0} & k_{i,j,1} & k_{i,j,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & k_{i,j,0} & k_{i,j,1} & k_{i,j,2} \end{bmatrix}.$$

Each row of $\mathbf{T}(\mathbf{K}_{i,j})$ corresponds to one spatial position of the convolution, and successive rows are obtained by shifting the kernel weights by S input indices, where S denotes the stride.

2. BTTB matrix assembly. Collecting all such Toeplitz matrices yields the Block-Toeplitz with Toeplitz Blocks (BTTB) matrix

$$\mathbf{T}_{\mathrm{BTTB}}(\mathbf{K}) \in \mathbb{R}^{24\times 30} = \begin{bmatrix} \mathbf{T}(\mathbf{K}_{0,0})\ \mathbf{T}(\mathbf{K}_{0,1})\ \mathbf{T}(\mathbf{K}_{0,2}) \\ \mathbf{T}(\mathbf{K}_{1,0})\ \mathbf{T}(\mathbf{K}_{1,1})\ \mathbf{T}(\mathbf{K}_{1,2}) \\ \mathbf{T}(\mathbf{K}_{2,0})\ \mathbf{T}(\mathbf{K}_{2,1})\ \mathbf{T}(\mathbf{K}_{2,2}) \end{bmatrix}.$$

3. Vectorized convolution. The input tensor $\mathbf{X} \in \mathbb{R}^{3\times 10}$ is vectorized in a channel-wise manner as

$$\mathrm{vec}(\mathbf{X}) = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \qquad \mathbf{x}_j = \begin{bmatrix} x_{j,0}\ x_{j,1} \cdots x_{j,9} \end{bmatrix}^\top, \quad j \in \{0,1,2\}.$$

The convolution can then be written as the linear mapping

$$\mathrm{vec}(\mathbf{Y}) = \mathbf{T}_{\mathrm{BTTB}}(\mathbf{K})\,\mathrm{vec}(\mathbf{X}).$$

This BTTB representation provides a unified linear-algebraic formulation of convolution, establishing a crucial theoretical bridge that extends geometric analysis frameworks originally developed for FCNs to CNNs. By transforming convolutional parameters into an equivalent fully connected weight matrix, the BTTB structure preserves the functional behavior of the original CNN while explicitly exposing the structured sparsity and repetition induced by parameter sharing. Under this reformulation, a convolutional layer can be viewed as a fully connected network with a highly structured weight matrix. All subsequent stages of our attack, including dual point finding, kernel-centric clustering, and parameter recovery, are then carried out entirely within this transformed FCN framework.

Crucially, while each row of $\mathbf{T}_{\mathrm{BTTB}}(\mathbf{K})$ corresponds to an equivalent neuron weight vector in an FCN formulation, these rows are not independent but are instead highly structured due to parameter sharing. Specifically, the rows corresponding to a single output feature map are all generated from a small, shared set of kernel. This inherent weight sharing is the key insight: it indicates that parameter recovery must operate at the *kernel level* rather than the individual *neuron level*. Building on this principle, the following section introduces a kernel-centric clustering algorithm.

3.3 Kernel-Centric Dual Point Clustering

In CNNs, the parameters of each convolutional kernel are repeatedly applied to the input space through parameter sharing. This mechanism is explicitly manifested in our proposed BTTB matrix representation. In this representation,

the rows corresponding to the weight vectors of different output neurons are simply spatially shifted replicas of the same convolutional kernel.Building on this structural property, we propose a kernel-centric dual point clustering strategy. In contrast to traditional neuron-wise matching, which requires an exhaustive pairwise comparison over all points, our approach identifies a sufficient set of dual points for each convolutional kernel individually. This method significantly reduces the search space and improves overall clustering efficiency.

To efficiently cluster the dual points while avoiding the $O(n^2)$ complexity of an all-pairs comparison, we propose the strategy detailed in Algorithm 1. The process takes a set of dual points D, a density threshold N, and a target number of clusters m as input. The algorithm operates by selecting a seed point p_s from the unclustered points to initialize a new cluster. This cluster is then populated by adding other points deemed consistent via the IsConsistent algorithm, which checks if the rank of their combined dual spaces is less than the input dimension d.Once two points are found consistent, the newly added point is immediately removed from D to prevent redundant comparisons and repeated clustering. The formation of a single cluster concludes once its size reaches the threshold N, ensuring it is sufficiently dense for robust signature recovery. This process repeats until all m target clusters are found.

After this initial clustering, a refine cluster step is applied to address potential false positives. Such errors can arise because the zero-padding inherent to the Toeplitz submatrix structure may produce spurious small singular values, leading to incorrect consistency judgments. Through this two-stage process of initial clustering and subsequent refinement, a final set of pure clusters is obtained and returned by the algorithm.

Determine the Cluster m via Coupon Collector's Problem Each convolutional kernel produces multiple identical weight vectors across output neurons due to parameter sharing. As a result, collecting m clusters is sufficient to probabilistically cover all convolutional kernels. Under the classical Coupon Collector's formulation, to guarantee that each of the C_{out} kernels is observed at least r times with probability at least $1-\delta$, the required number of clusters m satisfies

$$m \;\geq\; C_{\text{out}}\Big(\ln C_{\text{out}} + (r-1)\ln\ln C_{\text{out}} + \ln\frac{1}{\delta}\Big). \tag{10}$$

Here,

- C_{out} denotes the number of output channels, i.e., the number of convolutional kernels in the layer;
- r is the minimum number of times each kernel is required to be observed;
- δ is the allowable failure probability, meaning that the probability of at least one kernel being insufficiently covered is at most δ.

In our experiments on large-scale models, we set the minimum coverage requirement to $r = 3$ and choose $\delta = 0.02$, corresponding to a nominal success probability of $1 - \delta = 0.98$ under the idealized assumption that each kernel is

Algorithm 1 Kernel-Centric Cluster for CNNs

Input: Dual point set D, clustering threshold N, target number of clusters m
Output: Cluster set $\{C_1, C_2, \ldots, C_m\}$

Initialize cluster set $\mathcal{C} \leftarrow \emptyset$
while D is not empty **and** $|\mathcal{C}| < m$ **do**
  Select a seed p_s from D
  Initialize cluster $C \leftarrow \{p_s\}$
  for all $p \in D \setminus \{p_s\}$ **do**
    if IsConsistent(p, p_s) **then**
      $C \leftarrow C \cup \{p\}$
      $D \leftarrow D \setminus \{p\}$
    end if
    if $|C| \geq N$ **then**
      break
    end if
  end for
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
  $D \leftarrow D \setminus C$
end while
$\mathcal{C}_{\text{refined}} \leftarrow$ RefineClusters$(\mathcal{C})$
return $\mathcal{C}_{\text{refined}}$

detected with equal probability. This parameter choice is intentionally conservative.

We emphasize that the bound in Equation (10) is derived under a uniform sampling assumption, whereas in practical attack scenarios the distribution of collected dual points is generally non-uniform. Consequently, some convolutional kernels may be observed less frequently than predicted by the idealized model, and the formal probabilistic guarantee may no longer strictly apply.

To bridge this gap between theory and practice and to enhance the robustness of the overall attack, we adopt a conservative minimum coverage threshold of $r = 3$. This design serves two complementary purposes. First, it provides sufficient redundancy for the subsequent RefineClusters step, enabling effective identification and pruning of noisy or false-positive clusters, thereby improving cluster purity and parameter recovery accuracy. Second, while the 98% success probability should be interpreted as a nominal guideline rather than a strict guarantee under non-uniform sampling, this conservative setting ensures that the number of clusters m computed from Equation (10) remains empirically sufficient to achieve reliable coverage of all C_{out} convolutional kernels in practice.

Our kernel-centric clustering strategy incorporates two key optimizations for efficiency. First, the introduction of the clustering threshold N allows each clustering search to terminate early once a sufficient number of dual points is collected. This greatly improves empirical runtime while maintaining worst-case linear complexity. Second, we leverage the Coupon Collector's Problem to determine the target number of clusters m, ensuring that, with high probability,

each convolutional kernel is represented by at least one cluster. This probabilistic calculation enables efficient recovery without exhaustive neuron-wise matching.

To further enhance cluster purity, a RefineClusters step is applied after the initial clustering. In this stage, we randomly select triplets of dual points (a, b, c) from each cluster and verify their geometric consistency. Triplets that fail to conform to the characteristics of a single convolutional kernel are discarded. By iteratively removing inconsistent points, this refinement process reduces false positives and ensures that the final set of clusters reliably represents the underlying convolutional kernels.

3.4 Extract the Convolutional Layer

Following kernel-centric clustering, we proceed to recover the kernel signatures. This process begins by merging the dual spaces within each cluster to reconstruct its corresponding critical hyperplane. The normal vector of this hyperplane provides the weight vector for a single equivalent neuron, which in turn represents a convolutional kernel. For large-scale layers, this recovery is accelerated by applying SVD to the covariance matrix. The resulting weight vector inherently exhibits the expected sparsity and local continuity of an equivalent fully-connected representation. In the final step, this sparse vector is decomposed into its constituent kernel slices $\{K_{i,0}, K_{i,1}, \ldots, K_{i,C_{\text{in}}-1}\}$, to form the complete filter for a single output channel.

Unify Dual Spaces to Recover Signatures Following kernel-centric clustering, the weight recovery phase begins. The objective is to reconstruct the critical hyperplane for the neuron associated with each cluster. The intuition is that while each individual dual point corresponds to a $d-2$ dimensional local subspace, the neuron's entire critical hyperplane is their common extension. Therefore, by combining the dual spaces from all points within a single cluster, we use linear algebraic methods to approximate the unique $d-1$ dimensional hyperplane.

The normal vector of the reconstructed hyperplane constitutes the recovered weight vector. This vector is characterized by a distinct local structure: most of its components are zero, with only a few continuous non-zero regions. These non-zero segments are physically meaningful, as they directly correspond to the various channels of the convolutional kernel, while their collective position and length reflect the kernel's receptive field. Thus, the recovered vector not only defines the hyperplane's orientation but also inherently encodes the sparsity and local continuity of the underlying convolution operation.

However, a single cluster of dual points is often insufficient to uniquely determine this hyperplane. To resolve this and obtain a complete description of the neuron, it is necessary to find an another distinct cluster for the same neuron. This final stage is also grounded in linear algebra: we formulate the geometric information from each cluster as a system of linear equations and check for their

consistency. If the systems are consistent, their common solution yields the final, unambiguous critical hyperplane, thereby providing a full description of the neuron's parameters.

SVD-Based Acceleration Technique In our framework, assessing the geometric consistency of dual points relies on Singular Value Decomposition (SVD). The standard approach in Carlini *et al.* [3] performs a direct SVD on the full $n \times p$ data matrix, which becomes a bottleneck for large-scale networks when $n \gg p$ due to high memory and runtime costs. To address this, we introduce a two-fold acceleration strategy.

First, instead of direct SVD, we perform eigenvalue decomposition on the smaller $p \times p$ covariance matrix $C = D^\top D$, which is mathematically equivalent(see Appendix B for proof) and reduces both memory and runtime overhead in the high-sample regime. Second, C is computed incrementally via block-wise accumulation:

$$C = \sum_i (\text{batch}_i^\top \text{batch}_i), \tag{11}$$

so that memory depends only on batch size, allowing efficient processing of very large sample sets.

This two-fold strategy enables efficient handling of a larger number of dual samples while maintaining accuracy, significantly improving scalability and supporting the recovery of individual neuron weights.

Reconstruction of Convolutional Kernel Once a weight w for a single equivalent neuron is recovered, the goal is to reconstruct the original convolutional filter from it. This involves extracting the set of constituent kernel slices, $\{K_{i,0}, K_{i,1}, \ldots, K_{i,C_{\text{in}}-1}\}$, which collectively form the complete filter for a single output channel. The procedure effectively reverses the BTTB formulation: it decomposes the 1D sparse weight vector w, which represents a single row from the $T_{\text{BTTB}}(\mathbf{K})$ matrix, back into its underlying, structured kernel components.

The reconstruction process consists of the following steps:

1. **Decompose Neuron Weight $\boldsymbol{w}$.** The recovered weight vector $\boldsymbol{w}$, of length $C_{\text{in}} \cdot L_{\text{in}}$, is conceptually partitioned into C_{in} consecutive subvectors, $\{w_0, w_1, \ldots, w_{C_{\text{in}}-1}\}$, each of length L_{in}. Each subvector w_j contains the weights applied to the j-th input channel.
2. **Extract and Validate Kernel Slices.** From each subvector $\boldsymbol{w_j}$, the contiguous non-zero segment is extracted as a candidate kernel slice $K_{i,j}$. The set of candidate slices is then subjected to the following validity checks. If these checks do not pass, the recovered weight vector w is deemed invalid and discarded.
   - *Segment count:* w must contain exactly C_{in} non-zero segments.
   - *Segment length:* Each $K_{i,j}$ must have length equal to the kernel size K.

   Note: A segment shorter than K is normal at convolution boundaries where part of the kernel overlaps with padding.

3. **Reconstruction and Iteration.**
   - **Matrix Reconstruction:** From the recovered kernel slices $K_{i,j}$ and known layer parameters (K, S, P), we construct the corresponding Toeplitz submatrices $T(K_{i,j})$. This collection of submatrices forms the i-th block row of the full BTTB matrix, thereby providing a complete representation of the filter for the i-th output channel.
   - **Iterative Recovery:** After reconstructing one output-channel filter, select the next neuron cluster and repeat the extraction, validation, and reconstruction steps. Continue until all C_{out} output-channel filters are recovered, achieving a complete restoration of the convolutional layer parameters.

3.5 Extract the Average Pooling Layer

The attack relies on locating dual points near ReLU activations. In CNNs, average pooling layers are usually not followed by nonlinear activations (see Figure 1); thus, ReLU-based geometric methods cannot directly recover the average pooling layer weights. To address this limitation, we propose an analytical approach that leverages known architectural information (K, S, P) to deterministically reconstruct the weights of the average pooling layer. Since the K, S, and P of the average pooling layer are known from the architecture, its weights can be deterministically constructed without any geometric analysis. Once the correct K is identified by parameter verification, the average pooling weight is directly given by $1/K$. The attacker can therefore construct the sparse matrix representation of the pooling operation with full certainty.

Average pooling can be represented by a sparse matrix $T_{\text{pool}} \in \mathbb{R}^{L_{\text{out}} \times L_{\text{in}}}$, where each row encodes the linear averaging relation between input and output positions:

$$(T_{\text{pool}})_{r,c} = \begin{cases} \dfrac{1}{K}, & \text{if } rS - P \le c < rS - P + K, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Here $r = 0, 1, \ldots, L_{\text{out}} - 1$, and indices c outside $[0, L_{\text{in}} - 1]$ correspond to padding, whose contributions are zero.

If the input has C_{in} channels and pooling is applied independently to each channel, the complete layer mapping can be expressed as a block-diagonal matrix:

$$\mathbf{T}_{\text{BTTB}}(\mathbf{K}_{\text{pool}}) = \begin{bmatrix} T_{\text{pool}} & & \\ & \ddots & \\ & & T_{\text{pool}} \end{bmatrix} \in \mathbb{R}^{(C_{\text{in}} L_{\text{out}}) \times (C_{\text{in}} L_{\text{in}})}, \tag{13}$$

where each diagonal block represents the single-channel pooling transformation.

This reconstruction enables the attack to bypass the average pooling layer analytically: by multiplying the recovered outputs of preceding layers with the constructed pooling matrix, the attacker can compute the exact inputs to the

next layer, thereby restoring the conditions required to continue the geometric analysis recovery on deeper layers.

3.6 Sign Recovery

The repetitive structure of the BTTB matrix, a direct result of convolutional parameter sharing, allows us to reframe the sign recovery problem from the neuron level to the more fundamental kernel level. This reframing enables a more direct strategy: instead of processing each equivalent neuron individually, we perform an exhaustive search directly on the signs of the convolutional kernels. Since this set of kernels is typically small and manageable, this kernel-centric approach is sufficient to efficiently resolve the sign ambiguity for the entire layer.

4 Experiments

To validate the generalization and scalability of our approach, we conducted experiments on four representative CNNs, covering 1D and 2D convolutions and multi-channel, multi-kernel configurations. For evaluation on real datasets, we apply our method to the **LeNet-5** model trained on the **MNIST** dataset, a widely used benchmark for handwritten digit recognition. Our experiments show that the proposed recovery method successfully reconstructs LeNet-5, demonstrating its applicability to practical CNN architectures.

In the recovery process, the reconstruction is performed in a layer-by-layer manner. For each layer, we first conduct signature recovery and subsequently perform sign recovery, and this procedure continues until the entire network is fully reconstructed. All arithmetic operations are executed in 64-bit floating-point precision, and we assume that the parameters of all preceding layers have been perfectly extracted. The sign recovery for each neuron can be accomplished via exhaustive search.

4.1 Our CNN Networks

To evaluate the generality of the proposed signature recovery algorithm, we consider CNN architectures of different dimensionalities and complexities. Specifically, we include both 1D and 2D CNNs, extending to multi-channel and multi-kernel settings.

For the 1D case, we initially experimented with the real ECG5000 dataset. However, due to convergence issues that impeded the identification of decision boundaries, we instead adopted a synthetic dataset, which provides controlled conditions suitable for evaluating the 1D scenario.

For the 2D case, we use the classical **LeNet-5** model trained on the **MNIST** dataset. MNIST contains 60,000 training and 10,000 test grayscale images of handwritten digits (0–9), each of size $28 \times 28 \times 1$. The LeNet-5 architecture used in our experiments follows the standard configuration,consisting of the following sequence of convolutional, average pooling, and fully connected layers:

- Convolutional Layer 1: 6 filters of size 5×5, stride 1, padding 2, producing 6 feature maps of size 28×28.
- Average Pooling Layer 1: 2×2 pooling with stride 2, reducing the feature maps to size 14×14.
- Convolutional Layer 2: 16 filters of size 5×5, stride 1, producing 16 feature maps of size 10×10.
- Average Pooling Layer 2: 2×2 pooling with stride 2, reducing the maps to size 5×5.
- Convolutional Layer 3 (Flatten + FC Equivalent): 120 filters of size 5×5, resulting in a 120-dimensional feature vector.
- Fully Connected Layer 1: 120 input neurons, 84 output neurons, followed by ReLU activation.
- Fully Connected Layer 2 (Output Layer): 84 input neurons, 10 output neurons, followed by softmax activation.

The corresponding dimensional transformation of LeNet-5 can be summarized as:

$$\begin{aligned}(1,28,28) \to (6,28,28)_{\text{Conv}} \to (6,14,14)_{\text{Pool}} \to (16,10,10)_{\text{Conv}} \\ \to (16,5,5)_{\text{Pool}} \to 120 \to 84 \to 10.\end{aligned}$$

4.2 Signature Recovery Attack

With the theoretical foundations and core algorithms established in Section 3, we next present the experimental evaluation, detailing the behavior and performance of each component.

Description of the Implementation The total runtime for recovering the signatures of each target layer can be expressed as

$$t_{\text{signatures}} = t_{\text{dual}} + t_{\text{cluster}} + t_{\text{unify}}, \tag{14}$$

where t_{dual} denotes the time required to locate the *dual points*, t_{cluster} represents the runtime of the *kernel-centric clustering* procedure, which recovers all convolutional kernels in the target layer, and t_{unify} denotes the time required to reconstruct all kernels within the layer.

Our method not only accurately recovers the model parameters but also significantly reduces the overall recovery time compared to Carlini *et al.* approach [3]. The reduction in runtime mainly comes from the following two aspects:

1. **Kernel-Centric Clustering:** Our first optimization is a kernel-centric clustering strategy. Instead of clustering dual points for all neurons, only needs to identify one representative cluster for each unique convolutional kernel.This method avoids an exhaustive pairwise comparison over all dual points, leading to a significant reduction in the search space and a reduction in clustering time of up to 2^{10} seconds. Furthermore, by applying the Coupon Collector's Principle to determine the target number of clusters m, we can guarantee with high probability that all convolutional kernels are fully covered.

2. **SVD-Based Acceleration:** We employ an adaptive SVD-based acceleration strategy to overcome the poor scalability of the original algorithm. Our strategy intelligently adapts to the matrix shape at different stages of the recovery process:
   - During the initial clustering, the number of samples (n) is often only moderately larger than the feature dimension (p). In this regime, performing a direct SVD on the $n \times p$ data matrix is more computationally efficient because it avoids the overhead of calculating the covariance matrix. Therefore, we use the direct method for this phase.
   - Conversely, during the final recovery phase, the number of samples in the data matrix is much larger than the number of features ($n \gg p$). In this regime, we switch to the more efficient covariance matrix-based approach, which is both memory-efficient and faster.

Table 2. Experimental Results on CNNs.

Architecture (CNN–FCN)	**Conv Kernel (size × num)**	$\max \lvert\theta - \hat{\theta}\rvert$	**Queries**
80–49–18–10 (**240–49**–18–10)	32×1	$2^{-24.18}$	$2^{21.57}$
80–49–18–10 (**240–147**–18–10)	32×3	$2^{-24.83}$	$2^{23.34}$
784–**657–530**–403–10 (784–**657–530**–403–10)	128×1	$2^{-23.90}$	$2^{24.08}$
LeNet-5 (784–4704-**1176-1600**-400-120–84–10)	$(5, 5) \times 16$	$2^{-17.75}$	$2^{32.34}$

Bold layers in *Architecture* indicate the single layer being recovered.
$\max \lvert\theta - \hat{\theta}\rvert$: the maximum parameter error across all layers.
Queries: the number of queries used to recover the bold layer.

Table 2 presents the overall parameter recovery result of various CNN architectures after conversion to their equivalent FCN representations. Each row corresponds to a distinct network configuration, covering both one-dimensional and two-dimensional convolutional networks. As shown, our framework successfully reconstructs all parameters with a worst-case relative error of just $2^{-17.75}$ between the recovered and original values, indicating that they are nearly identical. Notably, even for practical large-scale architectures such as LeNet-5, which contain multiple convolutional and pooling layers as well as a substantial number of parameters, our method achieves highly accurate recovery, demonstrating strong scalability and robustness. Empirically, the query complexity is influenced by both the distribution of dual (critical) points and the input dimensionality.

Table 1 reports the signature recovery runtime for various CNN architectures, comparing Carlini *et al.* [3] and our method. To evaluate scalability and generalization, we test four representative CNNs, from the small 80–49–18–10 network to the large multi-channel LeNet-5, covering both 1D and 2D convolutions.

Although t_{dual} is computed similarly in both methods, the approach of Carlini *et al.* [3] recovers parameters independently for each neuron, causing runtime to grow rapidly with network size and making clustering slow without SVD acceleration. In contrast, our method uses a clustering threshold N and a target number of clusters m, enabling early termination and reducing both t_{cluster} and overall runtime.

Our kernel-centric clustering further improves efficiency: for the 784–657–530–403–10 network, the number of clusters drops from 530 to 1. Combined with SVD-based acceleration, our framework achieves speedups of up to $2^{3.8}$ for small networks and $2^{9.26}$ for medium-to-large networks. Notably, for complex models like LeNet-5, which the original method cannot recover in reasonable time, our approach successfully reconstructs all parameters within practical runtime.

5 Conclusions

This paper presents the first attack framework for extracting parameters of CNNs. To extend geometric attacks from FCNs to CNNs, we reformulate convolutional layers as sparse BTTB matrices and introduce two key optimizations, a kernel-centric clustering algorithm and an SVD-based acceleration strategy, addressing its scalability limitations. Consequently, these strategies enable efficient extraction across various CNN architectures, including multi-channel, multi-kernel, and average pooling configurations. To evaluate generalization and scalability, we conducted experiments on four representative CNN architectures. Our framework successfully reconstructs parameters, with a worst-case relative error of $2^{-17.75}$ between the recovered and original ones. In terms of efficiency, single-layer recovery achieves a speedup of approximately $2^{3.8}$ on small networks and up to $2^{9.26}$ on medium-scale networks, where speedups are more pronounced as network size increases. It is worth noting that we perform the first parameter recovery attack on the LeNet-5 model within a practical runtime. These results confirm the high accuracy and scalability of our method, validating its practical applicability for parameter extraction in modern CNNs.

In summary, this paper demonstrates the practical feasibility of fully reconstructing modern CNNs under highly restricted access conditions, posing a significant challenge to model confidentiality. A key component in many CNN architectures is the pooling layer, which typically comes in two common forms: average pooling and max pooling. By performing an analytical reconstruction based on known architectural information, our framework successfully handles average pooling layers. However, the non-linear nature of max pooling layers, which only retains the maximum value within each region, inherently leads to significant information loss and makes the operation irreversible. As a result, reconstructing the original input or parameters from these layers presents considerable difficulties, making the handling of such layers a persistent challenge for future research.

Acknowledgments. This research is supported by the Education Teaching Reform and Research Program of Shandong Province (Grant No. M2023243), the Education Teaching Reform and Research Program of Shandong University (Grant No. 2024Z23), the National Key R&D Program of China (Grant No. 2024YFA1013000, 2023YFA1009500), the National Natural Science Foundation of China (Grant No. 62032014, U2336207), and the National Cryptologic Science Fund of China (Grant No. 2025NCSF01013). Kai Hu is supported by National Cryptologic Science Fund of China(2025NCSF02007), the National Natural Science Foundation of China (62402283), Shandong Provincial Natural Science Foundation(No.2025HWYQ-025), the Natural Science Foundation of Jiangsu Province (BK20240420), Program of TaiShan Scholars Special Fund for young scholars (Grant NO.tsqn202507063) and Program of Qilu Young Scholars of Shandong University.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. 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
2. 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
3. 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
4. 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
5. 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

6. Coqueret, B., Carbone, M., Sentieys, O., Zaid, G.: A hard-label cryptanalytic extraction of non-fully connected deep neural networks using side-channel attacks. CoRR **abs/2411.10174** (2024). https://doi.org/10.48550/ARXIV.2411.10174
7. Daniely, A., Granot, E.: An exact poly-time membership-queries algorithm for extraction a three-layer ReLU network. CoRR **abs/2105.09673** (2021), https://arxiv.org/abs/2105.09673
8. Du, X.: Financial text analysis using 1D-CNN: Risk classification and auditing support. In: Proceedings of the 2025 International Conference on Artificial Intelligence and Computational Intelligence. pp. 515–520 (2025)
9. Fefferman, C., et al.: Reconstructing a neural net from its output. Revista Matemática Iberoamericana **10**(3), 507–556 (1994)
10. Ige, A.O., Sibiya, M.: State-of-the-art in 1D convolutional neural networks: A survey. IEEE Access **12**, 144082–144105 (2024). https://doi.org/10.1109/ACCESS.2024.3433513
11. 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
12. Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics, Springer, New York, NY, 2 edn. (2002). https://doi.org/10.1007/b98835
13. Kiranyaz, S., Ince, T., Gabbouj, M.: Real-time patient-specific ECG classification by 1-D convolutional neural networks. IEEE Trans. Biomed. Eng. **63**(3), 664–675 (2016). https://doi.org/10.1109/TBME.2015.2468589
14. Martinelli, F., Simsek, B., Gerstner, W., Brea, J.: Expand-and-cluster: Parameter recovery of neural networks. In: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net (2024), https://openreview.net/forum?id=3MIuPRJYwf
15. 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
16. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. CoRR **abs/1511.08458** (2015)
17. Serghini, O., Semlali, H., Maali, A., Ghammaz, A., Serrano, S.: 1-D convolutional neural network-based models for cooperative spectrum sensing. Future Internet **16**(1), 14 (2024). https://doi.org/10.3390/FI16010014

A IsConsistent Algorithm

Algorithm 2 IsConsistent(x_{dual0}, x_{dual1}, D_0, D_1)

Input: $x_{\text{dual}i}$a dual point of the model, S_i the corresponding dual space
Output: True if the two dual points are on the same neuron's critical hyperplane, otherwise False.

Let $\tilde{D}_j = f(x_{\text{dual}i}; i)(D_j)$
Let $D = \text{span}(\text{basis}(\tilde{D}_0) \cup \text{basis}(\tilde{D}_1))$
Let $X =$ the number of neurons active in either x_{dual0} or x_{dual1}.
if $\text{Rank}(D) = X$ **then**
    return False
else
    return True ▷ Specifically, $\text{Rank}(D) < X$

B Equivalence Between Covariance-Based Decomposition and Direct SVD

In this appendix, we formally establish the mathematical equivalence between performing a direct Singular Value Decomposition (SVD) on a data matrix and performing an eigenvalue decomposition on its covariance matrix. This result provides the theoretical foundation for the SVD-based acceleration strategy adopted in the parameter recovery stage of our framework. This equivalence is well established in the literature, see Jolliffe ([12, Section 3.5]) for a detailed treatment.

B.1 Problem Setup

Let $\mathbf{D} \in \mathbb{R}^{n \times p}$ denote the data matrix constructed, where n represents the number of samples and p denotes the dimensionality of the input feature space. During the final parameter recovery phase, a large number of samples are accumulated to ensure stable estimation, resulting in the high-sample regime $n \gg p$. The covariance-based decomposition introduced below is specifically designed to operate efficiently in this setting.

B.2 Method I: Direct Singular Value Decomposition

The standard approach performs an SVD directly on the data matrix $\mathbf{D}$:

$$\mathbf{D} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top, \tag{15}$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}$ and $\mathbf{V} \in \mathbb{R}^{p \times p}$ are orthogonal matrices whose columns are the left and right singular vectors, respectively, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times p}$ is a diagonal matrix containing the singular values.

This formulation is computationally efficient when n and p are of comparable scale and is therefore employed during the clustering stage of our algorithm.

B.3 Method II: Covariance Matrix Eigen-Decomposition

In the recovery stage, where $n \gg p$, we instead construct the covariance matrix

$$\mathbf{C} = \mathbf{D}^\top \mathbf{D} \in \mathbb{R}^{p \times p}. \tag{16}$$

Since $\mathbf{C}$ is symmetric and positive semi-definite, it admits an eigenvalue decomposition of the form

$$\mathbf{C} = \mathbf{P} \boldsymbol{\Lambda} \mathbf{P}^\top, \tag{17}$$

where the columns of $\mathbf{P}$ are the eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues.

B.4 Proof of Equivalence

Substituting the SVD of $\mathbf{D}$ into the definition of the covariance matrix yields

$$\begin{aligned} \mathbf{C} &= \mathbf{D}^\top \mathbf{D} \\ &= (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^\top)^\top (\mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^\top) \\ &= \mathbf{V} (\boldsymbol{\Sigma}^\top \boldsymbol{\Sigma}) \mathbf{V}^\top. \end{aligned} \tag{18}$$

The matrix $\boldsymbol{\Sigma}^\top \boldsymbol{\Sigma}$ is diagonal, with diagonal entries $\lambda_i = \sigma_i^2$, where σ_i are the singular values of $\mathbf{D}$. Defining

$$\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^\top \boldsymbol{\Sigma}, \tag{19}$$

we obtain

$$\mathbf{C} = \mathbf{V} \boldsymbol{\Lambda} \mathbf{V}^\top, \tag{20}$$

which is precisely the eigenvalue decomposition of the covariance matrix.

Therefore, the eigenvectors of $\mathbf{D}^\top \mathbf{D}$ coincide with the right singular vectors of $\mathbf{D}$, and the eigenvalues of the covariance matrix are the squares of the singular values of $\mathbf{D}$.

This equivalence shows that, in the recovery stage where $n \gg p$, performing an eigenvalue decomposition on the $p \times p$ covariance matrix yields the same geometric information as a direct SVD on the $n \times p$ data matrix, while significantly reducing memory and computational overhead.