ARCHITECTURE-AGNOSTIC HARD-LABEL CRYPTANALYTIC EXTRACTION OF CNNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Cryptanalytic model extraction attacks aim to recover the exact weights of a neural network given only black-box access and partial architectural knowledge, posing significant risks to model confidentiality and intellectual property. Prior cryptanalytic attacks against convolutional neural networks (CNNs) have demonstrated exact recovery in the hard-label setting, but require detailed architectural knowledge, including the number, size, and types of each layer, channel counts, kernel sizes, strides, and padding. These requirements limit their applicability in realistic black-box settings. In this work, we show that such detailed architectural knowledge is not necessary by considering a significantly weaker threat model in which the attacker does not know any additional architecture knowledge besides what is necessary to query the model. Under this threat model, we develop procedures to recover convolutional hyperparameters directly from extracted weight blocks, infer channel counts, account for sequential linear layers such as average pooling and batch normalization, and extract convolutional layers that are followed by max-pool operations. Our results demonstrate that exact cryptanalytic extraction of CNNs remains feasible in this architecture agnostic setting, narrowing the gap between prior cryptanalytic extraction attacks and practical black-box extraction settings.

1 INTRODUCTION

Model extraction attacks aim to recover the weights of a neural network given only black-box access, posing a significant risk to model confidentiality and intellectual property. Although the field originated in the 1990s (Fefferman et al., 1994; Blum & Rivest, 1988), many foundational concepts were first introduced by Tramèr et al. (2016), who proposed two distinct attack paradigms. Task accuracy extraction seeks an approximate model that performs well on the target task without reproducing the target model exactly. Fidelity extraction instead recovers the underlying parameters up to an equivalence class, producing identical outputs for every input.

Recent work has begun to apply parameter extraction attacks to CNNs (Sun et al., 2026; Chen et al., 2026a;b). However, these attacks require prior knowledge of several architectural details, including the type and shape of the target model's layers, and per-layer hyperparameters. The reliance on detailed architecture knowledge raises the natural question: is such architectural knowledge fundamentally necessary for exact CNN extraction? This paper answers this question by showing that detailed architectural knowledge assumed by previous CNN extraction methods can be recovered automatically within the existing pipeline, reducing the threat model significantly.

Contributions This paper shows that the architectural knowledge required by prior exact CNN extraction attacks is not necessary, and makes the following contributions:

- We develop procedures to infer kernel shape, stride, and padding directly from the extracted weight vectors, eliminating the need for prior knowledge of hyperparameter details.
- We show that the number of output channels in a convolution layer is determined by the total number of unique kernel solutions, removing the assumption of layer size.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- We show that multiple linear layers without intermediate ReLU activations can be extracted as one equivalent linear layer, allowing average pooling and batch normalization layers to be folded into an adjacent convolutional layer.
- We introduce methods to identify whether max pooling follows the target convolutional layer, and infer the pooling hyperparameters from the target layer's extracted ASVs.
- We demonstrate that exact cryptanalytic extraction remains possible under this weaker threat model across the 1D and 2D CNN architectures evaluated in this work.

2 Related Work

2.1 Cryptanalytic extraction of fully connected networks

Parameter extraction for fully connected networks (FCNs) was first formalized by Milli et al. (2019), whose approach required gradient access and assumed infinite-precision arithmetic. Jagielski et al. (2020) later produced a more practical variant by accounting for floating-point imprecision and estimating gradients via finite differences, although both results are limited to small, fully connected two-layer networks with ReLU activations.

Carlini et al. (2020) later reframed the problem through a cryptographic lens, enabling significant theoretical advances. Subsequent work optimized the attack (Canales-Martínez et al., 2024; Foerster et al., 2024) and extended it to the hard-label setting (Chen et al., 2024; Carlini et al., 2025; Canales-Martínez & Santos, 2025), where the API returns only a predicted class label rather than raw logits.

2.2 Cryptanalytic extraction of CNNs

Sun et al. (2026) proposed a method that converts CNN kernels into sparse matrices, enabling the existing hard-label attack to be applied directly to convolutional layers. Their approach further improves efficiency via kernel-centric dual-point clustering and SVD-based acceleration. Despite these contributions, the method does not support CNNs with max pooling layers and introduced additional architecture requirements: layer type, number of output channels, kernel size, padding, and stride.

Chen et al. (2026b) and Liu et al. (2026) both partially addressed the first limitation of Sun et al. (2026). These approaches introduce new types of critical points within max pooling layers to extract partial signatures and signs of preceding convolutional layers. However, both methods require access to raw logit outputs from the final layer. Recent improvements by Chen et al. (2026a) apply these methods to the hard-label setting, but full knowledge of the architecture remained a key assumption for the attack.

2.3 Architecture inference

Oh et al. (2019) trained metamodels to predict the architecture attributes from black-box models. Using this approach, their results are approximate with no parameter recovery.

Numerous approaches predict neural network architecture and parameters using side-channel attacks (Hu et al., 2020; Yan et al., 2020; Batina et al., 2019; Hua et al., 2018). While successful, the attacker requires additional access to the model beyond pure queries which adds an additional assumption to the threat model.

2.4 Copycat attacks

Before 2026, CNN model extraction research focused almost exclusively on task accuracy rather than fidelity (Correia-Silva et al., 2018; 2021). These copycat attacks query the target model to generate a synthetic dataset used to train an approximate model. Although effective, these attacks cannot recover functionally equivalent models.

Table 1: Threat models of cryptanalytic extraction attacks. "Knowledge" is the architectural knowledge the attacker is assumed to have.

Approach	Target	Output	Knowledge	Pooling
Carlini et al. (2020)	FC	Logits	Full	–
Canales-Martínez et al. (2024)	FC	Logits	Full	–
Carlini et al. (2025)	FC	Hard label	Full	–
Canales-Martínez & Santos (2025)	FC	Hard label	Full	–
Sun et al. (2026)	CNN	Hard label	Full + K/S/P	Avg
Liu et al. (2026)	CNN	Logits	Full + K/S/P	Max
Chen et al. (2026b)	CNN	Logits	Full + K/S/P	Max
Chen et al. (2026a)	CNN	Hard label	Full + K/S/P	Max
Ours	CNN	Hard label	None	Avg, Max

3 PRELIMINARIES

3.1 CNN DEFINITIONS

CNNs exploit local connectivity and parameter sharing to efficiently process image and sequential data. A typical CNN consists of alternating convolutional and pooling layers, followed by a flattening operation and one or more fully connected layers.

Definition 1 (Convolutional Layer) *A convolutional layer f_i is defined as $f_i : \mathbb{R}^{c_i \times h_i \times w_i} \to \mathbb{R}^{c_o \times h_o \times w_o}$ where $c_i, h_i, w_i, c_o, h_o, w_o \in \mathbb{Z}_+$, with $h_i \geq h_o$, $w_i \geq w_o$, and $N = c_o h_o w_o$. $f_i(x)$ applies a convolution with kernel size $K = (K_h, K_w) \in \mathbb{Z}^2_{\geq 0}$, stride $S = (S_h, S_w) \in \mathbb{Z}^2_+$, $S \leq K$, and padding $P = (P_h, P_w) \in \mathbb{Z}^2_{\geq 0}$, $P < K$.*

Definition 2 (Average Pooling Layer) *An average pooling layer is a convolutional layer with set weights applied to each channel independently $K_\theta(m, n) = 1/(K_h \times K_w)$.*

Definition 3 (Max Pooling Layer) *A max pooling layer partitions each channel of its input into windows and outputs the maximum value within each window.*

The final convolutional or pooling layer is followed by a flatten layer and one or more fully connected layers.

Definition 4 (Fully Connected Layer) *A fully connected layer $g_i : \mathbb{R}^I \to \mathbb{R}^N$ with weight matrix $A^{(i)}$ and bias $b^{(i)}$ performs the following affine transformation:*

$$g_i(x) = A^{(i)}x + b^{(i)}. \tag{1}$$

Convolutional and fully connected layers are usually followed by a nonlinear activation. In this paper, we exclusively study CNNs containing the ReLU activation function (Nair & Hinton, 2010) given by $\sigma(x) = \max(x, 0)$. While it may be possible to extend this attack to other activation functions (Qi et al., 2026; Asselineau et al., 2026), this remains future work.

We also consider batch normalization (Ioffe & Szegedy, 2015) in this paper. Batch normalization is represented as an affine transformation at inference time, similar to fully connected layers.

3.2 ATTACK BACKGROUND

Sun et al. (2026) extract convolutional layers by representing them as Block-Toeplitz with Toeplitz Blocks (BTTB) matrices. Because these matrices share the same structure as the weight matrices of fully connected layers, existing parameter extraction methods (Carlini et al., 2025; Canales-Martínez & Santos, 2025) can be applied directly.

To illustrate this conversion, consider a 2D convolution of a 3×3 input with a 2×2 kernel. This operation can equivalently be expressed as the matrix-vector product in Equation 2, where the kernel

entries populate a sparse Toeplitz submatrix as shown in Equation 3. The complete BTTB matrix is then assembled as a block matrix containing one such Toeplitz submatrix for each input-output channel pair.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \star \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{bmatrix} \tag{3}$$

The extraction procedure consists of three steps:

1. **Dual point collection.** Query the oracle to collect dual points consistent with the target layer.
2. **Clustering.** Group dual points by the kernel whose weights they correspond to.
3. **Kernel recovery.** Reconstruct each kernel's weights from its cluster of dual points.

Dual point collection and clustering remain unchanged from prior work (Carlini et al., 2025; Sun et al., 2026; Chen et al., 2026a). Kernel recovery, however, requires knowledge of the layer's hyperparameters and output shape, which we remove in Section 4.

3.2.1 DUAL POINTS

Definition 5 (Critical Point (Carlini et al., 2025)) *Let $\mathcal{V}(\eta; x)$ denote the pre-activation value of a hidden neuron η on input x. A critical point x satisfies $\mathcal{V}(\eta; x) = 0$ for some neuron η.*

Definition 6 (Decision Boundary (Chen et al., 2026a)) *The decision boundary consists of points so that two logit outputs of the model reach the same maximum value.*

Definition 7 (Dual Point (Carlini et al., 2025)) *A dual point d is a point that is both on the decision boundary and is a critical point for some neuron η.*

When a convolutional layer is followed by max pooling, critical points can be further categorized.

Definition 8 (ReLU-Pooling Critical Point, RPCP (Chen et al., 2026a)) *A RPCP is a critical point to some neuron η that is also the maximum value in a max pooling window.*

Definition 9 (Pooling Switching Point, PSP (Chen et al., 2026a)) *A PSP x consists of points such that two inputs within the local receptive field of a max pooling window reach the same maximum value.*

RPCPs (and PSPs) are dual RPCPs (and dual PSPs) if they also lie on a decision boundary. The dual point search algorithm can be found in Carlini et al. (2025), and the classification criteria for dual points in max-pooling models can be found in Chen et al. (2026a).

3.2.2 DUAL POINT CLUSTERING

Every dual point of the target layer lies on the critical hyperplane of a single neuron (or for dual PSPs, a pair of neurons) of that layer. Recovering the neuron's weights requires combining several dual points. Prior work clusters these dual points in one of two ways. Neuron-centric clustering groups dual points by neuron, while kernel-centric clustering groups dual points by output channel, combining multiple neuron-centric clusters at once. Kernel-centric clustering requires knowing where each neuron's kernel lies in the input, and therefore the layer's hyperparameters. Therefore, we use the neuron-centric Approximate Signature Vector (ASV) clustering, which requires no additional hyperparameter knowledge in this step. For each dual point, the ASV is computed from the normals of the decision boundary on both sides of the dual point. ASVs of dual points that belong to the same neuron are nearly parallel, so dual points are grouped by the direction of their ASVs. We refer to Chen et al. (2026a) for more details.

3.2.3 KERNEL RECOVERY

Kernel recovery begins by merging the dual spaces within each cluster to reconstruct the corresponding critical hyperplane. The normal vector of this hyperplane yields the weight vector of a single equivalent neuron, which encodes one convolutional kernel. This process is unchanged from prior work (Sun et al., 2026).

The recovered weight vector forms a row of the Toeplitz submatrix. Recovering the corresponding convolutional filter requires identifying contiguous nonzero blocks, concatenating them, and reshaping the result into a tensor of shape (c_i, K_h, K_w), where c_i is the number of input channels and K_h, K_w are the kernel height and width. This is repeated until all c_o unique kernels have been recovered. The layer can then be reconstructed from these kernels and the convolutional hyperparameters, completing the extraction.

We additionally recover the bias using the same method described in Chen et al. (2026a).

3.2.4 LAYER-BY-LAYER EXTRACTION

After extracting the first layer of a model, we can effectively "peel off" that layer to extract the next. Each additional layer requires significantly more dual points than the last, and is more prone to error propagation from prior extracted layers. Extracting the output layer of the model is implemented in Canales-Martínez & Santos (2025).

3.3 ADVERSARIAL GOALS AND ASSUMPTIONS

Under the threat model described above, the attacker can select queries x as inputs to an oracle $\mathcal{O}$, which returns the label of x produced by the CNN f_θ. The attacker's objective is to obtain the extracted parameters $\hat{\theta}$ which are functionally equivalent to the original parameters θ.

The following assumptions are made regarding the capabilities of the attacker:

1. **Full-domain inputs.** The attacker can query arbitrary inputs from the full input domain $\mathbb{R}^{d_0}$.
2. **Precise computations.** The CNN is specified and evaluated using a sufficiently high-precision floating-point arithmetic.
3. **Layer types.** The network is a sequential composition of convolutional, pooling, fully connected, and other linear layers such as batch normalization. The network uses only two nonlinear functions: the ReLU activation and max-pooling.

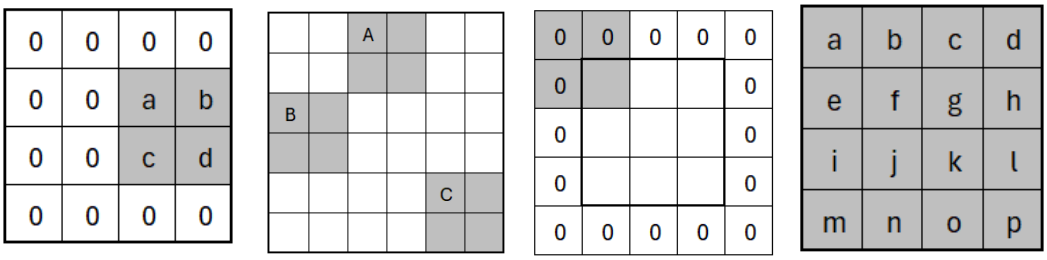
In this paper, we focus on layer-wise functionally equivalent extraction. Each layer is recovered given perfect knowledge of previous layers.

4 ARCHITECTURE-AGNOSTIC EXTRACTION

Using only knowledge of the input shape of the model, we show how to infer the remaining architectural information required for layer reconstruction. We assume that the weight vector for any given cluster is precisely extracted. Our goal is to recover the type, output channels, and hyperparameters of the target layer to complete the functionally equivalent extraction.

4.1 CONVOLUTIONAL HYPERPARAMETERS

After extracting the weight vector for a single output neuron, the previous approach uses known hyperparameters to reconstruct both the kernel and the full weight matrix for the kernel's corresponding output channel. To do so, the vector is split into contiguous nonzero blocks, representing the rows of the kernel itself. We are able to infer all of the hyperparameters by comparing the weight vectors between clusters.



(a) (b) (c) (d)

Figure 1: Four examples of extracted weight vectors projected onto a single input channel. **(a)** A 2x2 extracted kernel. **(b)** Three 2x2 extracted kernels with $S = 2$. **(c)** A 2x2 extracted kernel clipped by padding ($P = 1$). **(d)** A fully connected layer's extracted weight vector.

4.1.1 KERNEL

For convolutional layers, each recovered weight vector corresponds to one output neuron and therefore encodes a single convolutional kernel. Since the extracted weight vector is a row vector, we first reshape the vector into the layer's input shape as seen in Figure 1a.

The previous approach used the knowledge of the kernel's shape as a verification step to avoid information loss due to padding. Since we are extracting the weights from all clusters before processing, we only keep the kernels that share the maximum shape across all processed clusters.

4.1.2 STRIDE

The stride determines the spacing of adjacent kernel positions. To find the stride, we find the offset of each extracted kernel within its weight vector, then we calculate the relative difference between offsets, finally we find the greatest common factor of these differences.

Using Figure 1b as an example, kernel A is extracted at $(0, 2)$, B is extracted at $(2, 0)$, and C is extracted at $(4, 4)$. The differences between these offsets are $h_{\text{diff}} = \{2, 2\}$ and $w_{\text{diff}} = \{2, 4\}$. The greatest common factor for both height and width is 2, resulting in a stride of shape $(2, 2)$.

4.1.3 PADDING

For any extracted kernel that has a height or width smaller than the maximum, information was lost to padding. A visual representation of this effect is shown in Figure 1c where the extracted weight vector overlapped with the corner of a padded layer. To address this, we maintain lower bounds $\hat{P}_h$ and $\hat{P}_w$ on the padding. After processing all clusters, we can then set $P_h = \hat{P}_h$ and $P_w = \hat{P}_w$.

4.2 MAX POOLING

To account for max pooling layers, we must first classify clusters by their dual type. A PSP cluster will contain two copies of the same kernel with opposite signs, shifted by some displacement d. If at least two PSP clusters are identified, the model must contain a pooling layer. This threshold tolerates one misclassified cluster. All other clusters are then RPCP and may be used to extract kernel weights and convolutional hyperparameters.

4.2.1 POOLING KERNEL

Two neurons in the same pooling window are at most $K^P - 1$ output positions apart along each axis, and each output position corresponds to S input positions. The pooling kernel is therefore recovered from the largest displacement (d) observed across PSP clusters:

$$K_h^P = \max_{\text{PSP}} |d_h| \,/\, S_h + 1, \qquad K_w^P = \max_{\text{PSP}} |d_w| \,/\, S_w + 1. \tag{4}$$

4.2.2 POOLING STRIDE

Similar to convolutional stride, we take the gcd of the relative differences of the offsets from PSP clusters. Since the offsets can only differ by $S \times S^P$, we divide the gcd of differences by S to obtain S^P.

4.3 CHANNEL COUNT

The output channel count is equivalent to the number of uniquely extracted kernels. After processing all clusters, the extracted kernels are compared to find all unique solutions. Without knowledge of the layer's output size, this extraction step can propagate additional errors when insufficient dual points or clusters are found for the target layer.

4.4 MERGING LINEAR LAYERS

Since sequential linear layers with no intermediate ReLU activation can be merged into a single affine transformation, average pooling layers, batch normalization, and convolutional layers are all extracted at the same time to form one recovered layer that is functionally equivalent to the sequential layers of the target model. Therefore, for the purposes of this attack, we consider one layer to be a single weight-bearing linear layer that sits before one or more nonlinear layers (ReLU or max pooling).

4.5 FULLY CONNECTED LAYERS

Pre-ReLU fully connected layers are identified by an extracted weight vector that covers an entire input channel as shown in Figure 1d. Extracting intermediate fully connected layers without architecture knowledge is equivalent to extracting a convolutional layer with $K = (h_i, w_i)$, $S = 1$, $P = 0$ where the number of extracted kernels is approximately equivalent to the number of neurons in the target layer's output.

There are a few issues with this approach. Since each neuron is its own output channel, every neuron must be contained within a cluster, therefore persistent neurons, dead neurons, and near-dead neurons must be addressed. Persistent neurons are always positive ($\mathcal{V}(\eta; x) > 0$ and therefore cannot be matched to any dual point. Dead neurons are clipped by ReLU ($\mathcal{V}(\eta; x) \leq 0$. Finally, near-dead neurons are almost always dead and may require significantly more dual points to cluster. Since dead neurons do not affect the following layer of the model, an equivalent extraction can safely ignore them. We leave the extraction of persistent and near-dead neurons in the hard label setting without architecture knowledge to future work.

4.6 ARCHITECTURE EXTRACTION ALGORITHM

Algorithm 1 summarizes how a single layer is extracted without architecture knowledge. The attacker applies this algorithm once per layer and uses the previously extracted layers to solve for each cluster's weight vector in the input space of the target layer. The successfully solved weights are split into PSP and RPCP. RPCP weights are used to determine the convolutional hyperparameters and the set of unique kernels, while PSP weights are used to determine the pooling hyperparameters if at least 2 PSP clusters are found.

Without knowledge of the size of the target model, the attacker must iteratively increase the number of dual points until a solution is stable. After the architecture is recovered, the bias is extracted and the layer is returned. If multiple PSP clusters were found, a max pooling layer is appended to the result.

5 EXPERIMENTS

To validate our approach, we attack five representative CNNs spanning a range of architectural configurations. All experiments were conducted on a remote partition of a Dell PowerEdge XE8545 running Ubuntu 24.04 LTS with an AMD Milan 7713 CPU and a NVIDIA A100 SXM4 40GB GPU.

Algorithm 1 Architecture-agnostic extraction of a convolutional layer

Require: Oracle $\mathcal{O}$, previously extracted layers $\hat{f}_0 \dots \hat{f}_{i-1}$
Ensure: Extracted layer $\hat{f}_i$ (and $\hat{f}_{i+1}$ if Conv+Pool layer)
repeat
  $\mathcal{D} \leftarrow \text{CollectDuals}(\mathcal{O}, n)$
  $\mathcal{C} \leftarrow \text{Cluster}(\mathcal{D})$ ▷ neuron-centric; Sec. 3.2.2
  $W \leftarrow \{\text{Solve}(C, \hat{f}) : C \in \mathcal{C}\}$, dropping undetermined clusters
  Split W into RPCP and PSP weights ▷ Sec. 4.2
  $(K, S, P) \leftarrow \text{ConvParams(RPCP)}$ ▷ Secs. 4.1.1-4.1.3
  $(K^{\text{pool}}, S^{\text{pool}}) \leftarrow \text{PoolParams(PSP)}$ ▷ Sec. 4.2.1
  $\mathcal{K} \leftarrow$ unique kernels from RPCP clusters
  $c_o \leftarrow |\mathcal{K}|$ ▷ Sec. 4.3
  $n \leftarrow \alpha n$ ▷ $\alpha > 1$ to grow dual point budget
until solution is stable
Recover bias of $\mathcal{K}$
Build $\hat{f}_i$ from $\mathcal{K}$ and (K, S, P)
Build $\hat{f}_{i+1}$ from (K^P, S^P) if $|PSP| \geq 2$.
shape $\leftarrow$ output shape of the new layer
return $\hat{f}_i$ (and $\hat{f}_{i+1}$ if $|PSP| \geq 2$)

Table 2: Architecture recovery. The extracted type, channel count, convolutional and pooling hyperparameters, and whether all were extracted exactly. For FOLDED, layer 1 is functionally equivalent to a single convolution.

Model	L	Type	c_o	K	S	P	Pool (K, S)	Exact
1D	1	Conv	6	31	1	0	–	✓
	2	Conv	1	31	1	0	–	✓
FOLDED	1	Conv†	3	6	2	0	–	✓
NON-SQUARE	1	Conv	4	(3,5)	(1,2)	(0,1)	–	✓
	2	Conv	1	(5,3)	(2,1)	(1,1)	–	✗
MAX-POOL	1	Conv+Pool	1	3	1	0	(2,2)	✓
LeNet-5	1	Conv+Pool	6	5	1	2	(2,2)	✓
	2	Conv	1	5	1	0	–	✗

†Equivalent merged layer: AvgPool$(2, 2) \to$ Conv$(3, 1, 0) \to$ BN $=$ Conv$(6, 2, 0)$.

Our allocation contained 8 CPU cores with 30GB of RAM and a quarter partition of the A100. We used Python 3.12.3 with Pytorch to run and extract the CNNs.

The five target models are described in Appendix A. To reduce runtime, we compute decision-boundary normals from the model's gradients rather than estimating them from hard-label queries, as in (Chen et al., 2026a). Architecture recovery is exact if the layer type, channel count, convolutional hyperparameters, and pooling hyperparameters all match the ground truth labels. Weight and bias errors are the maximum absolute difference after rescaling with the ground truth kernel, since signature extraction recovers each neuron up to a scalar α. Neuron signs are taken from the ground truth. Each layer is recovered in isolation with perfect knowledge of previous layers. End-to-end extraction is left to future work.

5.1 Architecture Recovery

Table 2 reports the recovered architecture of each extracted layer. Of the 8 layers attacked, 6 were recovered exactly, with at least one per model. For non-square layer 2, no cluster contained the true padding height. For LeNet-5 layer 2, only one of the sixteen unique kernels was found since only 6 out of 23 clusters contained a valid solution. No PSP clusters were identified for this layer.

Table 3: Extraction fidelity and cost per layer. Coverage represents the number of successfully extracted kernels. Max error is the max absolute error after scaling to the ground-truth parameters. Number of duals is set by the attacker and iteratively increased until the solution is stable. Number of clusters is capped at 50, with an exception made for non-square layer 2, and is reported as RPCP/PSP for conv+pool layers. Time is reported in seconds for dual point collection, clustering, and recovery.

			Max error		Duals		Clusters		Rec.
Model	L	Coverage	Weights	Bias	#	(s)	#	(s)	(s)
1D	1	6/6	8.84×10^{-10}	3.54×10^{-8}	1K	108	50	4.1	57
	2	1/1	1.53×10^{-9}	4.05×10^{-8}	20K	2506	3	2.8	32
FOLDED	1	3/3	3.31×10^{-11}	2.72×10^{-8}	1K	128	46	4.8	190
NON-SQUARE	1	4/4	9.97×10^{-10}	4.16×10^{-7}	5K	382	49	22	222
	2	1/1	3.10×10^{-10}	4.19×10^{-8}	100K	8940	150	44	751
MAX-POOL	1	1/1	4.05×10^{-11}	4.71×10^{-8}	5K	581	9/41	23	227
LENET-5	1	6/6	7.88×10^{-10}	4.65×10^{-7}	50K	5823	15/26	443	174
	2	1/16	3.53×10^{-10}	8.22×10^{-8}	300K	33205	23	2552	75

5.2 PARAMETER EXTRACTION

Table 3 reports fidelity and cost per layer. Bias errors are consistently larger than weight errors, since the weight error propagates into the bias calculation. Dual point collection dominates the cost of deeper layers. In both model 1D and non-square, layer 2 required 20 times more dual points than layer 1, and about 23 times more time. In LeNet-5, collecting 300,000 dual points took over 9 hours of runtime. Clustering takes at most 45 seconds for every model except LeNet-5, which took over 40 minutes in layer 2. Kernel recovery, which includes architecture recovery, took at most a few minutes for all models but non-square, for which we raised the cluster cap to 150 because, at 50, most layer-2 dual points were not clustered.

6 CONCLUSIONS

Prior CNN extraction attacks assumed that detailed architecture knowledge was available to the attacker. We have shown that these assumptions are not required for functionally equivalent extraction in the hard-label setting. From the weight vectors that the existing attack already recovers, we infer kernel shape, stride, padding, channel count, and pooling hyperparameters. We also show that sequential linear layers without an intermediate ReLU can be extracted as one affine transformation. Across the 5 evaluated models, the architectures of 6 out of 8 layers were recovered exactly with low parameter error. These results suggest that keeping architecture information secret offers little protection against cryptanalytic extraction for CNNs with ReLU activations.

Limitations Our approach inherits assumptions of prior cryptanalytic attacks: full-domain inputs, precise computations, and limited options for nonlinear layers within the architecture. Beyond this, the extracted padding and channel count can be unreliable in deeper layers if the set of observed neurons is small. Padding is only revealed at the boundaries of the output space, which only occurs in a small fraction of each layer. Additionally full input channels can be missed if the required number of dual points is insufficient. This extends to fully connected layers, which often contain near-dead and persistent neurons in deeper layers preventing complete extraction of these layers. Finally, the dual budget grows quickly with layer depth, causing our evaluation to be limited to small networks. Scaling the attack to modern architectures with end-to-end extraction remains open.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on our experiments and implementing methods as code. We have not used generative AI tools for helping developing theoretical

models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, and interpreting results, and generating synthetic data sets, assisting with translations, clean and reformat dataset, and support qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, editing the research paper to improve readability, and proposing a title and keywords for the research paper. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

This work describes an attack that can be used to extract proprietary models from an API. All experiments were performed on models we trained ourselves with no queries to third-party models. The attack requires a large number of queries with high-precision inputs allowing for common countermeasures such as rate limiting and restricting the input space.

Reproducibility statement

Anonymized source code for the full attack, the target models and the evaluation scripts is provided in the supplementary material, together with a README describing how to reproduce every table. The target architectures are specified in Appendix A, and the hardware and software environment in Section 5. Ground truth parameters are used only to supply the prefix for the target layer and to compute metrics.

Acknowledgments

Redacted for anonymous review

References

Roderick Asselineau, Patrick Derbez, Pierre-Alain Fouque, and Brice Minaud. Cryptanalytic extraction of deep neural networks with non-linear activations. Cryptology ePrint Archive, Paper 2026/253, 2026. URL https://eprint.iacr.org/2026/253.

Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 515–532, Santa Clara, CA, August 2019. USENIX Association. ISBN 978-1-939133-06-9. URL https://www.usenix.org/conference/usenixsecurity19/presentation/batina.

Avrim Blum and Ronald Rivest. Training a 3-node neural network is np-complete. *Advances in neural information processing systems*, 1, 1988.

Isaac A Canales-Martínez and David Santos. Extracting some layers of deep neural networks in the hard-label setting. In *International Conference on Cryptology and Information Security in Latin America*, pp. 399–421. Springer, 2025.

Isaac A Canales-Martínez, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, Nitin Satpute, and Adi Shamir. Polynomial time cryptanalytic extraction of neural network models. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 3–33. Springer, 2024.

Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In *Annual international cryptology conference*, pp. 189–218. Springer, 2020.

Nicholas Carlini, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, and Adi Shamir. Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 364–396. Springer, 2025.

Yi Chen, Xiaoyang Dong, Jian Guo, Yantian Shen, Anyu Wang, and Xiaoyun Wang. Hard-label cryptanalytic extraction of neural network models. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 207–236. Springer, 2024.

Zirui Chen, Shi Tang, Zhengchao Gao, Yongjia Su, Lingyue Qin, and Xiaoyang Dong. Algebraic cryptanalytic extraction on hard-label neural networks, 2026a. URL https://arxiv.org/abs/2608.05736.

Zirui Chen, Shi Tang, Zhengchao Gao, Yongjia Su, Lingyue Qin, and Xiaoyang Dong. Algebraic attack on convolutional neural network with max pooling. Cryptology ePrint Archive, Paper 2026/241, 2026b. URL https://eprint.iacr.org/2026/241.

Jacson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2018.

Jacson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Copycat cnn: Are random non-labeled data enough to steal knowledge from black-box models? *Pattern Recognition*, 113:107830, 2021.

Charles Fefferman et al. Reconstructing a neural net from its output. *Revista Matemática Iberoamericana*, 10(3):507–556, 1994.

Hanna Foerster, Robert Mullins, Ilia Shumailov, and Jamie Hayes. Beyond slow signs in high-fidelity model extraction. *Advances in Neural Information Processing Systems*, 37:19496–19522, 2024.

Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pp. 385–399, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378460. URL https://doi.org/10.1145/3373376.3378460.

Weizhe Hua, Zhiru Zhang, and G. Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357005. doi: 10.1145/3195970.3196105. URL https://doi.org/10.1145/3195970.3196105.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security 20)*, pp. 1345–1362, 2020.

Haolin Liu, Adrien Siproudhis, Christina Boura, and Thomas Peyrin. Model extraction of convolutional neural networks with max-pooling. Cryptology ePrint Archive, Paper 2026/464, 2026. URL https://eprint.iacr.org/2026/464.

Smitha Milli, Ludwig Schmidt, Anca D Dragan, and Moritz Hardt. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pp. 1–9, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Seong Joon Oh, Bernt Schiele, and Mario Fritz. *Towards Reverse-Engineering Black-Box Neural Networks*, pp. 121–144. Springer International Publishing, Cham, 2019. ISBN 978-3-030-28954-6. doi: 10.1007/978-3-030-28954-6_7. URL https://doi.org/10.1007/978-3-030-28954-6_7.

Xiaokang Qi, Hao Lei, Longxiang Wei, Xiaohan Sun, and Meiqin Wang. Cryptanalytic extraction of neural networks with various activation functions. Cryptology ePrint Archive, Paper 2026/178, 2026. URL https://eprint.iacr.org/2026/178.

Xiaohan Sun, Hao Lei, Longxiang Wei, Xiaokang Qi, Kai Hu, Meiqin Wang, and Wei Wang. Cryptanalytic extraction of convolutional neural networks. Cryptology ePrint Archive, Paper 2026/139, 2026. URL https://eprint.iacr.org/2026/139.

Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pp. 601–618, 2016.

Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2003–2020. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/yan.

A Model architectures

The first model is 1D with small layer counts. The input shape is (3, 1, 80), and the final layer consists of 10 neurons.

1D 2 Conv1D layers followed by ReLU with $K = 31$, $S = 1$, $P = 0$. The first layer contains 6 output channels and the second contains 1.

The remaining four models use 2D convolutions with MNIST-shaped inputs (1, 28, 28). The final layer consists of 10 neurons.

folded One AvgPool2D with $S = K = 2$ followed by a Conv2D layer with $K = 3$, $S = 1$, $P = 0$ with 3 output channels followed by BatchNorm and a ReLU activation.

non-square 2 Conv2D layers each followed by ReLU. The first layer contains 4 output channels, $K = (3, 5)$, $S = (1, 2)$, and $P = (0, 1)$. The second layer converts the input into a square output with one output channel, $K = (5, 3)$, $S = (2, 1)$, and $P = (2, 1)$.

max-pool One Conv2D layer with $K = 3$, $S = 1$, $P = 0$ and 1 output channel followed by a ReLU then a MaxPool2D with $K = S = 2$.

LeNet-5 This is a complex architecture that mirrors the LeNet-5 architecture in (Sun et al., 2026) with max-pool instead of average-pool. The first layer is Conv2D with $c_o = 6$, $K = 5$, $S = 1$, $P = 2$ followed by ReLU and MaxPool with $K = S = 2$. The second layer is Conv2D with $c_o = 16$, $K = 5$, $S = 1$, $P = 0$ followed by ReLU and MaxPool with $K = S = 2$. The final two hidden layers are fully connected with 120 and 84 neurons respectively.