End-to-End Polynomial-Time Cryptanalytic Extraction of Convolutional Neural Networks in the Hard-Label Setting

Chun Li[1], Zheng Gong[1†], Di Li[1], Liping Zhuang[1], Yufeng Tang[2], Yin Lv[1], Xingfu Yan[1]
[1]School of Computer Science
South China Normal University
springli9807@gmail.com, cis.gong@gmail.com
edu.lidi97@gmail.com, 2025010261@m.scnu.edu.cn
lvyin@scnu.edu.cn, xfyan78@163.com
[2]Institute for Network Sciences and Cyberspace
Tsinghua University
yufengtang@tsinghua.edu.cn
[†]Corresponding author: Zheng Gong (cis.gong@gmail.com).

***Abstract*—Convolutional neural network parameters are valuable intellectual property, yet many APIs expose only top-1 labels and assume hidden logits limit parameter recovery. Prior cryptanalytic extraction can recover functionally equivalent ReLU MLPs, but CNNs introduce weight sharing, parallel critical hyperplanes, coupled spatial perturbations, and channel-sign ambiguity. This paper presents an end-to-end hard-label extraction attack for known-architecture ReLU CNN classifiers with average pooling. The main algorithmic contribution is channel-level recovery with SVGR-guided retained-candidate discrete optimization under a retained-candidate assumption. The attack locates dual points on decision and activation boundaries, recovers shared channel signatures with SVD, resolves channel signs, and peels layers while absorbing ReLU scale factors into later linear layers. Across evaluated 1D MNIST, 2D MNIST, and RGB CIFAR-10 variants, extraction reaches 100% prediction fidelity. Moreover, the evaluation demonstrates downstream security implications: extracted watermarked CNNs preserve behavior-level ownership evidence. Furthermore, the recovered models can be wrapped with deterministic triggers without erasing retained watermark signals, creating risk for both owners and downstream users. These results demonstrate that hiding logits alone does not protect parameters for this CNN family once architecture information is available. The code implementation is available for reproducibility.**

I. Introduction

Commercial model APIs commonly return only top-1 labels, with logits and confidence scores withheld to reduce model stealing risk. This interface is attractive because trained parameters encode costly data, architecture choices, and optimization effort that model owners treat as intellectual property [1]–[3]. The security question is whether hiding logits is enough to prevent parameter-level recovery when the model architecture is known.

Learning-based stealing usually targets task accuracy rather than parameter-level recovery [4], [5]. Cryptanalytic extraction instead targets a stronger property: parameter recovery or functionally equivalent behavior across the input space [5]–[7]. This paper studies functionally equivalent extraction under the finite-set condition in Section III. The core gap is end to end hard-label CNN extraction without reducing shared kernels to independent MLP neurons. Recent max-pooling CNN extraction attacks make progress under raw-output or soft-label access, where numeric outputs reveal local derivatives and pooling switches [8], [9]. These works solve an important but different problem from hard-label extraction. A max-pooling switch can change the selected hidden activation while leaving the top-1 class unchanged. The hard-label transcript then contains no reliable event marker for locating or classifying that switch.

A related security question is whether extracted CNNs preserve behavior-level watermarks and still support wrapper backdoor composition. Soft-label oracles expose logits or confidence scores that support continuous reconstruction techniques [6], [10]. Hard-label oracles expose only the predicted class [7], [11], removing the differential information used by prior soft-label attacks. The attacker adaptively queries a deterministic hard-label oracle with continuous probes and knows the architecture, following the cryptanalytic extraction threat model used in prior work [6], [7], [11]. This known-architecture assumption is explicit, and architecture recovery is a complementary problem outside this paper's contribution [12], [13]. The attack recovers channel signatures, resolves channel signs, and peels layers sequentially under this threat model. Figure 1 summarizes this four stage loop and its layer by layer application to target and extracted CNNs. The result demonstrates that hiding logits alone is insufficient for the known-architecture CNN family analyzed here. The evaluation also studies post-extraction impact.

Extracted copies can retain watermark behavior used for ownership evidence and can be wrapped with deterministic trigger logic. The central technical obstacle is that CNN weight sharing breaks the neuron-level isolation used by prior MLP

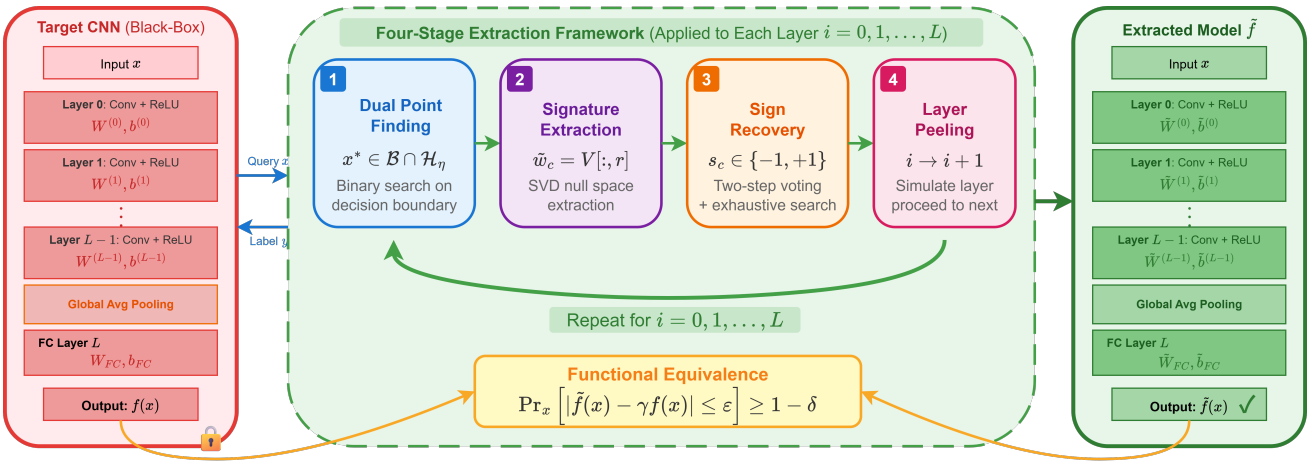

Fig. 1. Attack overview. Each layer repeats dual point finding, signature recovery, sign recovery, and layer peeling.

extraction methods.

A. Why Existing MLP Methods Fail on CNNs

Existing MLP extraction methods [6], [7], [10], [11], [14] rely on neuron level isolation for signature and sign recovery. CNN weight sharing breaks this assumption because one kernel creates parallel hyperplanes and coupled perturbation responses. Table I summarizes the resulting shift from neuron recovery to channel recovery.

Signature recovery fails due to parallel hyperplanes. MLP signature recovery uses dual points on distinct critical hyperplanes to recover each neuron's weight direction with SVD [6], [11]. In CNNs, all neurons in channel c share w_c, so their critical hyperplanes share a normal and differ only by spatial offsets. Dual points across these positions provide redundant channel evidence rather than independent neuron equations. The extraction target therefore shifts from neurons to channels.

Sign recovery fails due to coupled perturbations. MLP sign recovery perturbs inputs along one neuron's direction and observes asymmetric boundary distances [6], [10]. CNN weight sharing makes the same perturbation affect all overlapping positions in the channel. The response aggregates multiple neuron contributions and depends on receptive field overlap. Sign inference must therefore aggregate channel evidence instead of isolating one neuron.

TABLE I
MLP AND CNN EXTRACTION DIFFERENCES. WEIGHT SHARING SHIFTS RECOVERY FROM NEURONS TO CHANNELS.

Property	MLP	CNN
Weight structure	Independent	Shared within a channel
Critical hyperplanes	Distinct	Parallel within a channel
Perturbation effect	Single neuron	All spatial positions in the channel
Extraction target	Neuron	Channel

B. Pitfalls of Existing CNN Extraction Methods

Existing CNN extraction work must handle scale propagation, layer chaining, and channel-sign search under shared kernels. Raw-output and soft-label max-pooling attacks infer local affine changes from numeric output variation [8], [9]. A top-1 oracle suppresses this signal because a pooling switch can change the selected hidden activation without changing the predicted class. Sun et al. study hard-label CNN extraction with BTTB representations and kernel-centric clustering, but sign recovery remains tied to an unresolved sign space [15]. These pitfalls motivate channel-level recovery and SVGR-guided retained-candidate sign optimization for the average-pooling CNN family studied here.

TABLE II
CRYPTANALYTIC EXTRACTION COMPARISON. SIGNATURE AND SIGN RECOVERY ARE REPORTED SEPARATELY.

Network Architecture					Parameter Recovery		Reference
Type	Dataset	Architecture	Neurons	Oracle	Signature	Sign	
MLP	MNIST [16]	784-128-1	128	Raw	$\mathcal{P}$	$\mathcal{E}$	[5]
MLP	Arbitrary [17]	10-20-20-1	40	Raw	$\mathcal{P}$	$\mathcal{E}$	[17]
MLP	MNIST [16]	10-20-20-1	40	Raw	$\mathcal{P}$	$\mathcal{E}$	[6]
MLP	MNIST	784-128-1	128	Raw	$\mathcal{P}$	$\mathcal{E}$	[6]
MLP	CIFAR-10	3072-256×8-10	2,048	Raw	$\mathcal{P}$	$\mathcal{P}$	[10]
MLP	MNIST, CIFAR-10	100-50-50-1	100	Raw	$\mathcal{P}$	$\mathcal{P}$	[18]
MLP	MNIST, CIFAR-10	1024-2-2-1	4	HL	$\mathcal{P}$	$\mathcal{E}$	[7]
MLP	CIFAR-10	3072-256×3-64-10	832	HL	$\mathcal{P}$	$\mathcal{P}$	[11]
CNN	MNIST	LeNet(6-16-120)-FC(10)	6,518	HL	$\mathcal{P}$	$\mathcal{E}$	[15]
CNN	MNIST	MP-LeNet(6-16)-FC(120-84-10)	6,518	Raw	$\mathcal{P}$	$\mathcal{P}$	[8]
CNN	CIFAR-10	MP-Conv(3-4)-FC(256-120-10)	4,482	Raw	$\mathcal{P}$	$\mathcal{P}$	[8]
CNN	CIFAR-10	MP-VGG(4-8-8-16)-FC(16-10)	9,242	Soft	$\mathcal{P}$	$\mathcal{P}$	[9]
CNN	**MNIST**	**Conv(8-8-4)-FC(10)**	**5,096**	**HL**	$\mathcal{P}$	$\mathcal{P}$	This work
CNN	**CIFAR-10**	**LeNet(6-16-120)-FC(10)**	**8,662**	**HL**	$\mathcal{P}$	$\mathcal{P}$	This work

Notes: 'Architecture' gives input-hidden-output dimensions for MLPs and Conv(channels)-FC(outputs) for CNNs. MP denotes max pooling, and FC denotes fully connected. Raw denotes complete numeric outputs, Soft denotes soft-label or pre-normalization numeric outputs, and HL denotes hard-label top-1 access. $\mathcal{P}$ denotes polynomial time under each work's stated assumptions, and $\mathcal{E}$ denotes exponential time.

C. Contributions

Table II positions the paper's end-to-end hard-label CNN extraction attack against prior cryptanalytic extraction attacks. Under the stated candidate filtering assumption, the attack targets functionally equivalent extraction for deterministic, known-architecture CNN classifiers with ReLU activations and

average pooling. This hard-label oracle model differs from prior max-pooling CNN extraction under raw-output or soft-label access [8], [9]. The attack is built from four contributions:

- **Channel-oriented signature recovery.** The attack replaces neuron-level recovery with channel-level SVD constraints. Weight sharing turns parallel spatial hyperplanes into redundant evidence, allowing the attack to recover channel weight directions across the tested CNN variants.
- **Two-step sign recovery.** The attack aggregates asymmetric boundary-distance votes across spatial positions to handle coupled channel perturbations. SVGR-guided discrete optimization resolves retained low-confidence signs within a polynomially bounded candidate set.
- **Layer-peeling propagation and evaluation.** Layer peeling propagates recovered signs and signatures layer by layer while absorbing unknown scales into later linear layers. Across multiple evaluated structures, extraction reaches 100% prediction fidelity under hard-label access.
- **Security-implication evaluation.** The evaluation demonstrates that stolen watermarked models may still carry owner provenance evidence, while attackers can add wrapper triggers without erasing that evidence. This composition creates a harder abuse case: theft remains attributable to the owner-facing watermark, yet the derivative service can exhibit new trigger-controlled behavior.

These recovery techniques combine into an end-to-end cryptanalytic extraction attack for convolutional neural networks.

The rest of the paper follows the first-level section structure. Section II reviews learning-based and cryptanalytic extraction work. Section III defines the oracle model, network model, and extraction goal. Section IV presents the CNN extraction pipeline and its complexity analysis. Section V evaluates extraction fidelity, query cost, scalability, and security implications. Section VI discusses assumptions, limits, and defensive implications, and Section VII concludes.

II. Related Work

A. Learning-based model stealing attacks

Learning-based attacks query the target model to label a dataset and then train a substitute model on the resulting input-label pairs. Since Tramer et al. [4], this line of work has improved query selection, data generation, and distillation-based copying [2], [19]–[22].

These methods are useful for task-level mimicry, but they do not target parameter recovery or exact decision-boundary agreement. Random initialization, data-order stochasticity, and optimizer choices affect training trajectories and final models [23], [24]. Independently trained networks can reach similar accuracy through different parameterizations and internal representations [25]–[27]. Jagielski et al. [5] demonstrate that high-fidelity matching is difficult even with access to the original training data and network architecture.

B. Cryptanalytic extraction attacks

Cryptanalytic extraction attacks treat prediction APIs as parameter oracles rather than labeling services. Fefferman et al. [28] established identifiability results for neural networks under complete input to output behavior. Jagielski et al. [5] highlighted the gap between high accuracy substitutes and stronger fidelity goals. Carlini et al. [6] and Shamir et al. [10] made this goal algorithmic for ReLU MLPs using numeric outputs and local linear structure. Foerster et al. [18] further studied sign recovery under related cryptanalytic extraction settings.

Hard-label extraction removes the numeric signal used by soft-label attacks. Chen et al. [7] used boundary points to replace logits for restricted ReLU networks, but their method targets small binary classifiers and retains exponential sign recovery. Carlini et al. [11] later gave polynomial-time hard-label extraction for deep fully connected networks under their stated assumptions. These works demonstrate that top-1 labels can leak MLP parameters, but they do not address convolutional weight sharing.

CNN extraction requires channel-level reasoning because shared kernels create parallel critical hyperplanes. Chen et al. and Liu et al. address max-pooling CNNs under raw-output and soft-label access [8], [9]. Those attacks use numeric output variation to identify affine changes or pooling switches, which hard-label oracles hide behind a class index. A pooling switch may change internal activation without changing the final class, so max-pooling needs additional hard-label observability conditions. Sun et al. use BTTB representations and kernel-centric clustering, but sign resolution can grow with the unresolved kernel-sign space [15]. This paper instead uses channel-level SVD constraints and SVGR-guided retained-candidate sign optimization under hard-label access, assuming no side-channel leakage [1], [12], [13], [29].

III. Preliminaries

The following notation states the CNN extraction problem. The definitions cover the target architecture, geometric objects, and hard-label threat model. For clarity, the derivation uses flattened 1D notation. The 2D and RGB experiments use the same operators with tensor-indexing changes only. For quick reference, a notation summary is provided in Appendix A.

Definition 1 (Convolutional Neural Network)**.** *A convolutional neural network is a function $f : \mathbb{R}^{d_0} \to \mathbb{R}^K$. Eq. 1 defines the network composition.*

$$f = f_{FC} \circ f_{pool} \circ f_{L_{conv}-1} \circ \cdots \circ f_1 \circ f_0 \tag{1}$$

Here $f_0, \ldots, f_{L_{conv}-1}$ are convolutional layers, f_{pool} is a fixed linear average-pooling operator, and $f_{FC} : \mathbb{R}^{d_{pool}} \to \mathbb{R}^K$ is the fully connected classifier. The complete extraction pipeline has $L = L_{conv} + 1$ extraction layers when the FC layer is counted.

The input $x \in \mathbb{R}^{d_0}$ propagates through L_{conv} convolutional layers. Each layer applies convolution, adds bias, and applies ReLU activation. The final pooling and FC layers produce K class logits.

Definition 2 (Convolutional Layer)**.** *A convolutional layer f_i transforms input h to output activations. Eq. 2 defines the layer transformation.*

$$f_i(h) = \sigma\left(A^{(i)} \cdot G_i(h) + b^{(i)}\right) \tag{2}$$

Here $A^{(i)} \in \mathbb{R}^{C_{out}^{(i)} \times (C_{in}^{(i)} \cdot k^{(i)})}$ is the weight matrix, $b^{(i)} \in \mathbb{R}^{C_{out}^{(i)}}$ is the bias vector, $G_i(\cdot)$ is the patch extraction operator, and $\sigma(\cdot)$ is the ReLU activation.

Layer i has $C_{\text{in}}^{(i)}$ input channels and $C_{\text{out}}^{(i)}$ output channels. The kernel size is $k^{(i)}$ with stride $s^{(i)}$. Each output channel corresponds to one row of the weight matrix $A^{(i)}$.

Definition 3 (Patch Extraction Operator)**.** *The patch extraction operator G_i implements the im2col transformation. Eq. 3 defines the patch extraction transformation.*

$$G_i(h)[:, p] = vec\left(h[:, p \cdot s^{(i)} : p \cdot s^{(i)} + k^{(i)}]\right) \tag{3}$$

Here $p \in \{0, 1, \ldots, n_{patches}^{(i)} - 1\}$ indexes spatial positions and $n_{patches}^{(i)} = \lfloor (d_{i-1} - k^{(i)})/s^{(i)} \rfloor + 1$.

Each column of $G_i(h)$ contains a flattened receptive field patch.

Definition 4 (ReLU Activation)**.** *The ReLU activation function applies elementwise nonlinearity. Eq. 4 defines the ReLU activation.*

$$\sigma(z) = \max(0, z) \tag{4}$$

Here z is the pre-activation input and the output is zero for negative inputs.

Definition 5 (Linear Average Pooling)**.** *Linear average pooling f_{pool} reduces spatial dimensions by averaging over fixed pooling regions. Eq. 5 defines the pooling operation.*

$$f_{pool}(h)[c, q] = \frac{1}{|R_q|} \sum_{p \in R_q} h[c, p] \tag{5}$$

Here $h \in \mathbb{R}^{C_{out}^{(L_{conv}-1)} \times n_{patches}^{(L_{conv}-1)}}$ is the final convolutional layer output, c indexes output channels, and $\{R_q\}$ is a fixed collection of pooling regions. Global average pooling is the special case with one region per channel.

Definition 6 (Fully-Connected Layer)**.** *The fully-connected layer f_{FC} maps pooled features to class logits. The mapping is defined by Eq. 6.*

$$f_{FC}(\bar{h}) = W_{FC} \cdot \bar{h} + b_{FC} \tag{6}$$

Here $W_{FC} \in \mathbb{R}^{K \times d_{pool}}$ is the weight matrix, $b_{FC} \in \mathbb{R}^K$ is the bias vector, and $\bar{h} \in \mathbb{R}^{d_{pool}}$ is the vectorized pooled feature representation.

The extraction attack exploits geometric structures induced by ReLU activations. The key geometric objects are neurons, critical hyperplanes, decision boundaries, and dual points.

Definition 7 (Neuron)**.** *A neuron η is indexed by a tuple which specifies its location in the network. The neuron index is defined by Eq. 7.*

$$\eta = (i, c, p) \tag{7}$$

Here $i \in \{0, \ldots, L_{conv} - 1\}$ is the convolutional layer index, $c \in \{0, \ldots, C_{out}^{(i)} - 1\}$ is the channel index, and $p \in \{0, \ldots, n_{patches}^{(i)} - 1\}$ is the spatial position.

Definition 8 (Pre-activation)**.** *The pre-activation value determines whether a neuron is active. The pre-activation of neuron $\eta = (i, c, p)$ at input x is defined by Eq. 8.*

$$z(\eta, x) = w_c^{(i)\top} \cdot G_i(h^{(i-1)})[:, p] + b_c^{(i)} \tag{8}$$

Here $w_c^{(i)}$ is the weight vector for channel c, $h^{(i-1)} = f_{i-1} \circ \cdots \circ f_0(x)$ is the layer input activation, and $b_c^{(i)}$ is the bias.

The neuron is active when $z(\eta, x) > 0$ and inactive when $z(\eta, x) < 0$. The transition occurs at the critical hyperplane where $z(\eta, x) = 0$. Weight sharing implies neurons at different positions share parameters: $w_c^{(i)}$ and $b_c^{(i)}$ are identical for all p.

Definition 9 (Critical Hyperplane)**.** *The critical hyperplane of neuron η separates its active and inactive regions. The hyperplane is defined by Eq. 9.*

$$\mathcal{H}_\eta = \left\{x \in \mathbb{R}^{d_0} : z(\eta, x) = 0\right\} \tag{9}$$

Here $z(\eta, x)$ is the pre-activation value, and the set contains all inputs where the neuron transitions between active and inactive states.

The hyperplane has codimension 1 in input space. Points on $\mathcal{H}_\eta$ satisfy the linear constraint $w_c^{(i)\top} \cdot G_i(h^{(i-1)})[:, p] + b_c^{(i)} = 0$. The weight vector $w_c^{(i)}$ is normal to this hyperplane in the local coordinate system.

Definition 10 (Decision Boundary)**.** *The decision boundary between classes j and ℓ separates their prediction regions. The boundary is defined by Eq. 10.*

$$\mathcal{B}_{j,\ell} = \{x \in \mathbb{R}^{d_0} : f(x)[j] = f(x)[\ell]\} \tag{10}$$

Here $f(x)[j]$ and $f(x)[\ell]$ are the logits for classes j and ℓ, respectively.

Definition 11 (Dual Point)**.** *A dual point x^* lies on both the decision boundary and a critical hyperplane. The dual point condition is defined by Eq. 11.*

$$x^* \in \mathcal{B}_{j,\ell} \cap \mathcal{H}_\eta \tag{11}$$

Here $\mathcal{B}_{j,\ell}$ is the decision boundary between classes j and ℓ, and $\mathcal{H}_\eta$ is the critical hyperplane of neuron η.

Dual points provide geometric constraints for weight recovery. At a dual point, both $g(x^*) = 0$ and $z(\eta, x^*) = 0$ hold simultaneously. The intersection forms a $(d_0 - 2)$-dimensional affine subspace generically.

The paper considers a black-box setting with hard-label oracle access. The attacker queries the model and observes only the predicted class.

Definition 12 (Hard-Label Oracle)**.** *The hard-label oracle induced by a logit network returns only the predicted class label without confidence scores. The oracle function is defined by Eq. 12.*

$$\mathcal{O}_g(x) = \operatorname*{argmax}_{j\in[K]} g(x)[j] \tag{12}$$

Here g is any logit network, and ties are broken by selecting the smallest index j. In the attack, $\mathcal{O}_f$ is the queried target oracle.

The attacker in the hard-label setting has no access to logits, gradients, or internal activations. The goal is high-fidelity parameter recovery with as few hard-label queries as possible.

Definition 13 (Functionally Equivalent Extraction)**.** *Functionally equivalent extraction aims to generate an extracted model $\tilde{f}_{\hat{\theta}}$ whose logits match the target up to positive scaling on an evaluation set. Eq. 13 formalizes this finite-set success condition.*

$$\Pr_{x\in S}\left[\left\|\tilde{f}_{\hat{\theta}}(x) - \gamma f_\theta(x)\right\|_\infty \le \varepsilon\right] \ge 1-\delta. \tag{13}$$

Here $\gamma > 0$ is the scaling constant, S is the evaluation set, ε is the logit tolerance, and δ is the allowed failure probability. A positive γ preserves the hard-label oracle in Definition 12, while PF later measures sampled label agreement as a weaker empirical check.

Scale ambiguity from ReLU homogeneity prevents exact raw weight equality. These per-channel scale factors are absorbed into later layers so that the final logits satisfy the positive scale condition in Definition 13.

Definition 14 (Signature)**.** *The signature of channel c captures the weight direction without magnitude. The signature is defined by Eq. 14.*

$$\tilde{w}_c = \frac{w_c}{\|w_c\|_2} \in \mathbb{S}^{d_w-1} \tag{14}$$

Here $w_c \in \mathbb{R}^{d_w}$ is the true weight vector and $\mathbb{S}^{d_w-1}$ is the unit sphere.

IV. Cryptanalytic Extraction of CNNs

The attack targets the known-architecture CNN family defined in Section III. This threat model follows cryptanalytic extraction work, where the attacker uses chosen inputs to recover model parameters or a functionally equivalent model [6], [7], [11].

A. Threat Model and Scope

- **Oracle access.** The attacker adaptively queries a deterministic hard-label oracle $\mathcal{O}_f$ and observes only the predicted class.
- **Query space.** Queries may be continuous probes in $\mathbb{R}^{d_0}$, including points generated by interpolation, boundary walking, and local perturbation.
- **Architecture knowledge.** The attacker knows layer dimensions, channel counts, kernel sizes, strides, pooling type, and output classes.
- **Model scope.** The analysis covers deterministic CNN classifiers with ReLU activations, average pooling, and classification outputs.

B. Attack Overview

The attack turns hard-label queries into boundary and dual-point constraints, then peels layers sequentially. Each extracted layer repeats four stages:

- **Dual Point Finding.** This stage locates input points that simultaneously lie on a decision boundary and a neuron's critical hyperplane. These dual points provide geometric constraints for weight recovery.
- **Signature Recovery.** SVD is applied to dual point constraints to recover each channel's weight direction. Weight sharing provides redundant constraints for numerical stability.
- **Sign Recovery.** Sign ambiguity is resolved through boundary-distance votes aggregated across spatial positions. SVGR-guided discrete optimization handles the retained low-confidence channel candidates.
- **Layer Peeling.** The recovered layer maps later probes into the next representation. This stage propagates recovered signs and scales while later linear layers absorb remaining ReLU scale factors.

This four-stage loop reduces deep CNN extraction to repeated layer-local recovery under the stated assumptions.

C. Dual Point Finding

Dual point finding searches for inputs lying simultaneously on the decision boundary and a neuron's critical hyperplane [6], [11]. These dual points provide geometric constraints for weight reconstruction. The procedure localizes a boundary point via binary search, estimates its normal with finite differences, and walks tangentially until a neuron switches activation. The following subsections define these steps.

1) Decision Boundary Localization: The first step in dual point discovery localizes points on the decision boundary. Binary search between two inputs with different predicted classes finds boundary crossings by halving the interval at each iteration.

Let $x_0, x_1 \in \mathbb{R}^{d_0}$ be two points satisfying $\mathcal{O}_f(x_0) \neq \mathcal{O}_f(x_1)$, where $\mathcal{O}_f$ is the hard-label oracle from Definition 12. By continuity of the logits, the line segment crosses at least one decision boundary. In multiclass settings, midpoint bisection records the labels on the final bracketing endpoints and uses that adjacent class pair for subsequent constraints. Midpoint bisection iteratively narrows the interval containing the boundary crossing.

Initialize $y_0^{(0)} = x_0$ and $y_1^{(0)} = x_1$. The midpoint at iteration t is defined by Eq. 15.

$$m^{(t)} = \frac{y_0^{(t)} + y_1^{(t)}}{2}. \tag{15}$$

Here $y_0^{(t)}$ and $y_1^{(t)}$ are the interval endpoints at iteration t. Query the oracle to obtain $\mathcal{O}_f(m^{(t)})$. Update the interval endpoints based on the query result. If $\mathcal{O}_f(m^{(t)}) = \mathcal{O}_f(x_0)$, the boundary lies between $m^{(t)}$ and $y_1^{(t)}$, so the update sets $y_0^{(t+1)} = m^{(t)}$ and $y_1^{(t+1)} = y_1^{(t)}$. Otherwise, set $y_0^{(t+1)} = y_0^{(t)}$ and $y_1^{(t+1)} = m^{(t)}$. This maintains the invariant $\mathcal{O}_f(y_0^{(t)}) = \mathcal{O}_f(x_0)$ and $\mathcal{O}_f(y_1^{(t)}) = \mathcal{O}_f(x_1)$ for all t.

The interval length decreases exponentially. Eq. 16 gives the convergence rate.

$$\|y_0^{(t)} - y_1^{(t)}\| = \frac{1}{2^t}\|x_0 - x_1\| \tag{16}$$

Here t is the number of iterations and $\|x_0 - x_1\|$ is the initial interval length. Precision ε requires $T = \lceil \log_2(\|x_0 - x_1\|/\varepsilon) \rceil$ iterations. Each iteration consumes exactly one oracle query, yielding logarithmic query complexity. The final point $y_0^{(T)}$ satisfies $\|y_0^{(T)} - x_b\| \leq \varepsilon$ for some true boundary point x_b.

In the implementation, $\varepsilon = 10^{-16}$ approaches floating-point precision. For inputs in $\mathbb{R}^{784}$ with unit norm, this typically requires 50-100 queries depending on the initial separation $\|x_0 - x_1\|$. The resulting boundary approximation is denoted x_b and used as the starting point for boundary walking.

2) Boundary Direction Estimation: After locating a boundary point x_b, the algorithm estimates the normal vector $n_b \in \mathbb{R}^{d_0}$ to the decision boundary. Without gradients in the hard-label setting, n_b is estimated using finite differences along basis directions $\{v_1, \ldots, v_\kappa\}$ in a chosen subspace. For each direction v_j, the algorithm measures the forward and backward distances δ_j^+ and δ_j^- from x_b to the decision boundary. The ratio of these distances approximates the directional derivative of the gap function.

Aggregating contributions from all κ basis directions, the estimated normal vector is represented in Eq. 17.

$$\hat{n}_b = \sum_{j=1}^{\kappa} \frac{\delta_j^- - \delta_j^+}{\delta_j^- + \delta_j^+} v_j. \tag{17}$$

Here $\hat{n}_b$ is the unnormalized normal estimate computed from all κ basis directions. The estimate is normalized to obtain a unit normal $n_b = \hat{n}_b / \|\hat{n}_b\|_2$. The detailed derivation and computational procedure are provided in Appendix B-A.

3) Boundary Walking for Dual Point Locating: Given a boundary point x_b and its normal n_b, tangential boundary walking discovers dual points. A random direction is first projected onto the tangent space to obtain a walk direction $\hat{t}_b$ perpendicular to the normal. Walking along $\hat{t}_b$ maintains proximity to the boundary until the local geometry changes because a neuron switches activation state.

The algorithm iteratively steps along $\hat{t}_b$ while testing whether the current position remains on the decision boundary. When the boundary proximity test fails, the location is treated as a candidate transition where a neuron activation changes the boundary geometry. Binary search then precisely locates the dual point x_{dual} at this transition.

For each discovered dual point, the algorithm stores a triple $(x_{\text{mid}}, x_{\text{dual}}, x_{\text{end}})$ containing the dual point and its left/right context points. These triples are used for subsequent sign recovery and weight reconstruction. The detailed mathematical formulation is provided in Appendix B-B.

The complete dual point finding procedure is presented as Algorithm 2 in Appendix C-B. The algorithm takes as input the hard-label oracle $\mathcal{O}_f$, the desired number of dual points m, the subspace dimension κ, and the precision parameter ε. It outputs a collection of dual point triples $\{(x_{\text{left}}^{(i)}, x_{\text{dual}}^{(i)}, x_{\text{right}}^{(i)})\}_{i=1}^{m}$ suitable for subsequent weight reconstruction.

D. Channel-oriented Signature Recovery

Prior work on MLP extraction applies SVD to dual point constraints, recovering one weight vector per neuron [6]. In CNNs, weight sharing means neurons at different spatial positions share identical weights. Dual point clusters first correspond to spatial neuron or path units. Clusters sharing a channel are then aggregated for shared kernel SVD. SVD then recovers channel-level weight signatures from the null-space structure.

Figure 2 illustrates the signature extraction process for both convolutional and fully-connected layers. The process consists of three stages. First, dual points are collected and organized into spatial path clusters. The G operator then extracts patches to construct the constraint matrix. Finally, channel aggregation and SVD recover shared weight signatures from the null space.

1) Dual Point Clustering: The main clustering challenge is organizing collected dual points by the channel or neuron that induced them. A dual point x^* lies simultaneously on the decision boundary and on a neuron's critical hyperplane, forming a $(d_0 - 2)$-dimensional dual space D^η at their intersection. Two dual points from the same neuron η yield dual spaces whose union remains rank-deficient, while dual points from different neurons produce full-rank unions. This rank criterion supports clustering through consistency testing: for dual spaces D_i and D_j, if $\text{rank}(D_i \cup D_j) < d_{\text{active}}$ where d_{active} counts non-dead neurons, the points are consistent and are assigned to the same source.

The clustering procedure constructs a consistency graph where nodes represent dual points and edges connect consistent pairs. Maximal clique detection identifies candidate source-specific clusters and filters spurious edges caused by numerical instability. These clusters are spatial neuron or path units before channel aggregation. Clusters mapped to the same output channel provide constraints for one shared kernel. Each cluster provides multiple constraint equations for the same weight vector, enabling overdetermined signature extraction.

2) Convolutional Layer Signature Recovery: For convolutional layer $i \in \{0, \ldots, L_{\text{conv}} - 1\}$, the algorithm constructs a constraint matrix from dual points and extracts the weight signature via SVD. The input layer ($i = 0$) operates directly on raw input x^*, while hidden layers ($i > 0$) require forward propagation through previously extracted parameters $\{\tilde{A}^{(j)}, \tilde{b}^{(j)}\}_{j=0}^{i-1}$.

a) Constraint Formulation.: Let $h^{(i-1)}$ denote the layer i input activation. For $i = 0$, $h^{(-1)} = x^*$. For $i > 0$, $h^{(i-1)} =$

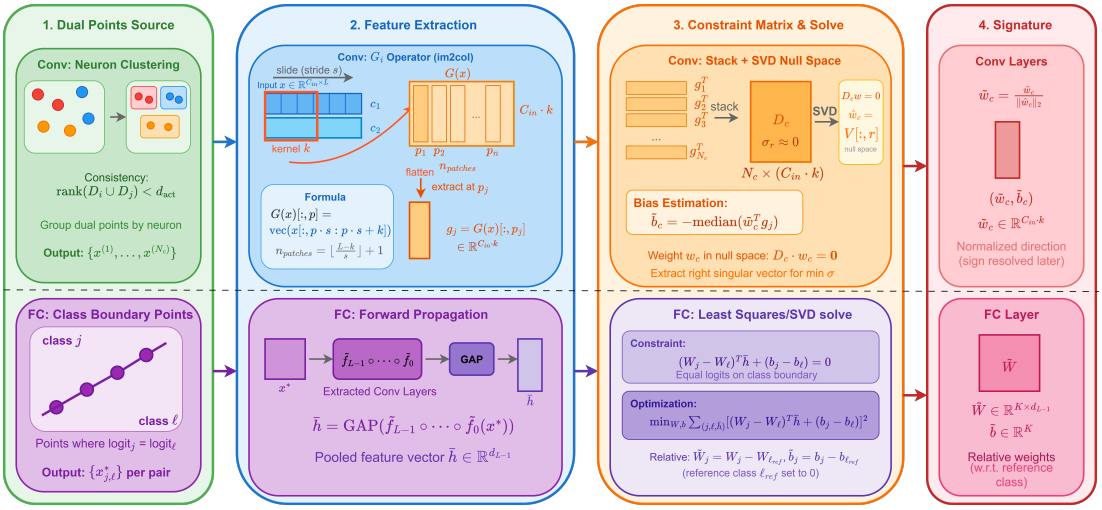

Fig. 2. Signature extraction. Convolutional layers use channel SVD, and FC layers use least squares.

$\tilde{f}_{i-1} \circ \cdots \circ \tilde{f}_0(x^*)$. Each dual point x^* for channel c at spatial position p satisfies a linear constraint. The critical hyperplane equation is defined by Eq. 18.

$$w_c^\top \cdot G_i(h^{(i-1)})[:, p] + b_c = 0 \tag{18}$$

Here $w_c \in \mathbb{R}^{C_{\text{in}}^{(i)} \cdot k^{(i)}}$ is the weight vector, and $G_i(\cdot)[:, p]$ extracts the patch at position p.

For channel c, let $\{x^{(1)}, \ldots, x^{(N_c)}\}$ denote N_c collected dual points with spatial positions $\{p_1, \ldots, p_{N_c}\}$. The rows are centered within each channel before SVD, which removes the shared bias term in Eq. 18. Let $\bar{g}_c$ denote the mean patch vector over the channel constraints. The constraint matrix is defined by Eq. 19.

$$D_c = \begin{bmatrix} \left(G_i(h^{(1)})[:, p_1] - \bar{g}_c\right)^\top \\ \left(G_i(h^{(2)})[:, p_2] - \bar{g}_c\right)^\top \\ \vdots \\ \left(G_i(h^{(N_c)})[:, p_{N_c}] - \bar{g}_c\right)^\top \end{bmatrix} \tag{19}$$

Here $h^{(j)}$ is the propagated activation for dual point $x^{(j)}$, and centering cancels the constant bias across rows.

b) SVD-Based Extraction.: After centering, the weight vector w_c lies in the null space of D_c, satisfying $D_c \cdot w_c = \mathbf{0}$. Singular value decomposition $D_c = U \Sigma V^\top$ extracts the right singular vector corresponding to the minimal singular value. The extracted weight signature is defined by Eq. 20.

$$\tilde{w}_c = \frac{V[:, r]}{\|V[:, r]\|_2} \tag{20}$$

Here $V[:, r]$ is the right singular vector associated with ς_r. The sign of $\tilde{w}_c$ remains ambiguous and is resolved in the subsequent sign recovery phase. Weight sharing reduces the number of distinct channel signatures from $C_{\text{out}} \times n_{\text{patches}}$ position-specific targets to C_{out} channel-level targets.

c) Bias Estimation.: The bias is estimated using median aggregation over dual point constraints. The bias estimation is defined by Eq. 21.

$$\tilde{b}_c = -\text{median}\left(\{\tilde{w}_c^\top \cdot G_i(h^{(j)})[:, p_j]\}_{j=1}^{N_c}\right) \tag{21}$$

The median reduces sensitivity to outliers and accumulated numerical errors.

3) Output Layer Signature Recovery: The output layer is a fully connected classifier mapping pooled features to class logits. The extraction requires all convolutional layer parameters $\{\tilde{A}^{(j)}, \tilde{b}^{(j)}\}_{j=0}^{L_{\text{conv}}-1}$ to be already recovered.

a) Constraint Formulation.: The pooled feature vector $\bar{h}$ is computed by forward propagating dual points through all extracted convolutional layers followed by the target average-pooling operator. Dual points on the decision boundary between classes j and ℓ satisfy the constraint defined by Eq. 22.

$$(W_j - W_\ell)^\top \bar{h} + (b_j - b_\ell) = 0 \tag{22}$$

Here $W_j, W_\ell \in \mathbb{R}^d$ are weight vectors and b_j, b_ℓ are biases for classes j and ℓ.

b) Least Squares Extraction.: To solve the constraint system, a reference class ℓ_{ref} is introduced with relative weights $\tilde{W}_j = W_j - W_{\ell_{\text{ref}}}$ and $\tilde{b}_j = b_j - b_{\ell_{\text{ref}}}$. The overdetermined system is solved by least squares as defined in Eq. 23.

$$(\tilde{W}, \tilde{b}) = \arg\min_{W,b} \sum_{(j,\ell,\bar{h})} \left[(W_j - W_\ell)^\top \bar{h} + (b_j - b_\ell)\right]^2 \tag{23}$$

The sum ranges over all collected dual points together with their class-pair labels and pooled features. The bias terms are recovered jointly with weights, with the reference class bias set to zero.

The recovered signatures $\{\tilde{w}_c\}$ are recovered up to per-channel sign and scale ambiguities. Appendix C-C gives the procedures in Algorithms 3/4/5 for the input, hidden, and output layers, respectively.

E. Two-Step Sign Recovery

The neuron wiggle technique [6], [10] determines sign by measuring asymmetric boundary distances, but assumes independent perturbation of individual neurons. In CNNs, weight sharing couples all spatial positions: perturbing along w_c affects all neurons in channel c simultaneously. The attack addresses this with two-step sign recovery combining position-wise voting and SVGR-guided discrete optimization over retained candidates. SVGR, defined as the ratio between the smallest and second-smallest singular values $\varsigma_{\min}/\varsigma_{\min+1}$, serves as the extraction quality metric during candidate verification.

Sign ambiguity problem. SVD-based signature recovery extracts weight directions but not their orientations. Both w_c and $-w_c$ produce identical null space solutions. This ambiguity arises because a hyperplane's zero set is unchanged by simultaneous sign reversal.

The critical hyperplane $w_c^\top x + b_c = 0$ defines the same zero set as $(-w_c)^\top x + (-b_c) = 0$. The orientation still matters because it determines which side of the hyperplane is active under ReLU. The recovery procedure therefore must infer one orientation per output channel.

For layer i, the total number of sign configurations is defined by Eq. 24.

$$|\mathcal{S}^{(i)}| = 2^{C_{\text{out}}^{(i)}} \tag{24}$$

Here $\mathcal{S}^{(i)}$ denotes the sign configuration space for layer i, $C_{\text{out}}^{(i)}$ is the number of output channels, and each channel $c \in \{1, \ldots, C_{\text{out}}^{(i)}\}$ contributes a binary choice $s_c \in \{-1, +1\}$.

1) Neuron Wiggle: The neuron wiggle method [6] determines sign by measuring asymmetric distances to the decision boundary. In MLP extraction, each neuron is perturbed independently. In CNN extraction, weight sharing requires aggregating measurements across all spatial positions within a channel.

Figure 3 illustrates the distance asymmetry principle. The critical hyperplane $\mathcal{H}_\eta$ separates the on-side (neuron active) from the off-side (neuron inactive). Perturbations from a dual point x^* toward the on-side reach the next decision boundary more quickly than perturbations toward the off-side. This asymmetry provides evidence for the weight-sign orientation.

Consider a point x near the critical hyperplane of neuron η with extracted weight direction $\tilde{w}$. The pre-activation value is $z = \tilde{w}^\top G_i(h)[:, p] + \tilde{b}$ where h is the layer input and $G_i(\cdot)$ extracts the patch at position p. The distance from x to the decision boundary depends on which side of the hyperplane x occupies.

The on-side is the region where the neuron is active ($z > 0$), and the off-side is the inactive region ($z < 0$). Let Δ_{on} denote the distance from x to the boundary when perturbing toward the on-side. Let Δ_{off} denote the distance when perturbing toward the off-side.

The key observation is the asymmetry between these distances. The distance asymmetry is defined by Eq. 25.

$$\Delta_{\text{on}} < \Delta_{\text{off}} \quad \text{when } s_c = +1 \tag{25}$$

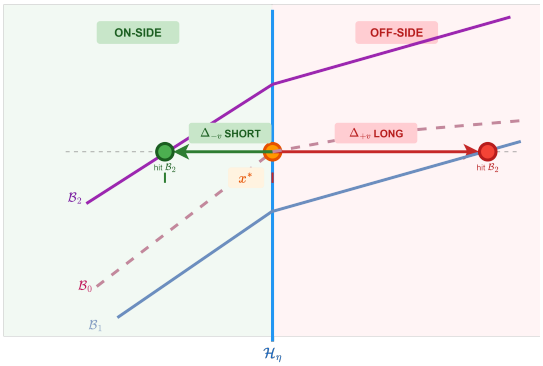

Fig. 3. Distance asymmetry for sign recovery. Boundary distances differ across the two perturbation directions.

Here s_c denotes the true sign of channel c. When the extracted sign matches the true sign, perturbations toward the on-side reach the boundary more quickly. This asymmetry provides a discriminative signal for sign determination.

The perturbation direction for channel c at position p maps the weight signature to input space. The input-space perturbation is defined by Eq. 26.

$$v_{c,p} = \mathcal{P}_{i \to 0}(\tilde{w}_c, p) \tag{26}$$

Here $\mathcal{P}_{i \to 0}$ denotes the projection from layer i to input space. For the input layer ($i = 0$), this projection directly unfolds the kernel weights into input coordinates. For hidden layers ($i > 0$), this projection requires backpropagation through extracted parameters.

For test points x_0 near the boundary, the hard-label oracle is queried with bidirectional perturbations $x_0 \pm \epsilon v_{c,p}$. The response from the indicator function is defined by Eq. 27.

$$\Delta(x_0, v, \epsilon) = \mathbb{1}\left[\mathcal{O}_f(x_0 + \epsilon v) \neq \mathcal{O}_f(x_0)\right] \tag{27}$$

Here $\mathcal{O}_f$ is the hard-label oracle and $\mathbb{1}[\cdot]$ is the indicator function. A response of 1 indicates that the perturbation crossed a decision boundary.

The position-level vote aggregates responses across multiple test points as defined in Eq. 28.

$$\nu_{c,p} = \sum_{x_0, \epsilon} \left[\Delta(x_0, v_{c,p}, \epsilon) - \Delta(x_0, -v_{c,p}, \epsilon)\right] \tag{28}$$

The sum ranges over dual points x_0 and perturbation magnitudes ϵ. A positive vote indicates that positive perturbations cross boundaries more frequently. This asymmetry provides the position-level vote for sign orientation.

2) Two-Step Sign Recovery: Sign recovery decomposes into two complementary steps. The first step applies voting-based determination using the neuron wiggle method. The second step solves a SVGR-guided discrete optimization problem over the ambiguous channels retained by the confidence filter.

Figure 4 presents the two-step workflow. Step 1 applies neuron wiggle voting across all spatial positions and classifies channels by confidence. Channels with high confidence are accepted directly. Channels with low confidence enter Step

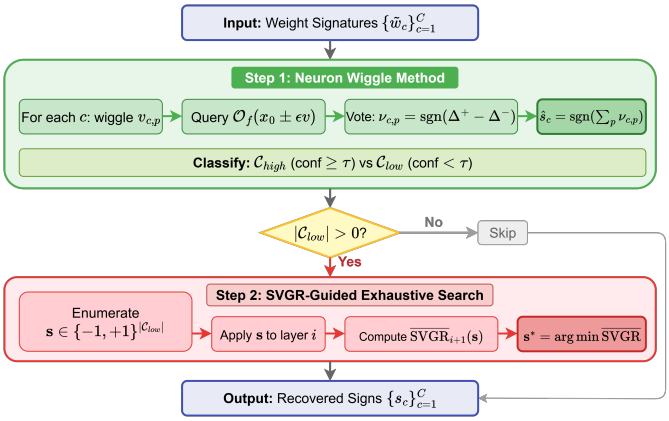

Fig. 4. Two-step sign recovery. Voting filters channels before SVGR-guided discrete optimization.

2, which optimizes over their retained sign configurations. Candidate verification minimizes next-layer SVGR for non-final convolutional layers and minimizes the FC least-squares residual for the final convolutional layer. The optimization stage considers sign configurations only for ambiguous channels. The configuration with the best layer-specific verification score is selected for that layer. The implementation uses a confidence metric to reduce the number of channels entering this optimization stage.

3) Convolutional Layer Sign Recovery: For convolutional layer $i \in \{0, \ldots, L_{\text{conv}} - 1\}$, the algorithm applies the two-step procedure with a layer-specific verification objective. The input layer ($i = 0$) operates directly on raw input, while hidden layers ($i > 0$) require Jacobian backpropagation through previously extracted parameters.

a) Voting Stage.: The input-space perturbation direction is defined by Eq. 29.

$$v_{c,p}^{(i)} = \begin{cases} \text{unfold}(\tilde{w}_c, p) & i = 0 \\ \left[\mathcal{J}^{(i)}(x_0)\right]^\top \cdot \text{unfold}(\tilde{w}_c, p) & i > 0 \end{cases} \tag{29}$$

Here $\text{unfold}(\cdot, p)$ places kernel weights at the receptive field position p, and $\mathcal{J}^{(i)}$ is the Jacobian of layers 0 through $i-1$.

The channel-level sign is estimated by aggregating position votes, as defined by Eq. 30.

$$\hat{s}_c^{(i)} = \text{sign}\left(\sum_p \nu_{c,p}^{(i)}\right) \tag{30}$$

Here $\nu_{c,p}^{(i)}$ is the position vote from Eq. 28. Channels with strong vote margins are accepted, and weak margins enter discrete optimization.

b) Discrete Candidate Verification.: For the $|\mathcal{C}_{\text{low}}|$ low-confidence channels, a discrete minimization problem ranges over $2^{|\mathcal{C}_{\text{low}}|}$ retained sign assignments. The signed parameters are $\tilde{A}_c^{(i)} = s_c \cdot \tilde{w}_c$ and $\tilde{b}_c^{(i)} = s_c \cdot \tilde{b}_c$.

The selected configuration minimizes the verification objective in Eq. 31.

$$\mathbf{s}^{(i)*} = \begin{cases} \text{argmin}_{\mathbf{s}}\, \text{SVGR}_{i+1}(\mathbf{s}), & i < L_{\text{conv}} - 1, \\ \text{argmin}_{\mathbf{s}}\, \text{FCRes}(\mathbf{s}), & i = L_{\text{conv}} - 1. \end{cases} \tag{31}$$

Here $\mathbf{s}$ ranges over retained assignments in $\{-1,+1\}^{|\mathcal{C}_{\text{low}}|}$, SVGR_{i+1} denotes mean next-layer SVGR, and FCRes denotes the least-squares residual of the FC extraction system in Eq. 23. Thus, the final convolutional layer uses FC solvability, while the FC layer's remaining global sign is resolved by prediction fidelity.

4) Sign Recovery for Output Layer: The fully-connected output layer has only a global sign ambiguity after relative weight extraction. The relative weights $W_j - W_{\ell_{\text{ref}}}$ are defined up to an overall sign flip. End-to-end prediction fidelity resolves this ambiguity.

The extraction produces weights $(\tilde{W}_{\text{FC}}, \tilde{b}_{\text{FC}})$ with ambiguous global sign. Both sign choices $s \in \{-1, +1\}$ are evaluated on a test set. The prediction fidelity for each sign is defined by Eq. 32.

$$\text{PF}_s = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \mathbb{1}\left[\mathcal{O}_f(x_j) = \mathcal{O}_{\tilde{f}_s}(x_j)\right] \tag{32}$$

Here $\tilde{f}_s$ denotes the extracted logit network with global FC sign s.

The global sign s_{FC}^* is selected by maximizing prediction fidelity as defined in Eq. 33.

$$s_{\text{FC}}^* = \underset{s \in \{-1,+1\}}{\text{argmax}}\ \text{PF}_s \tag{33}$$

Here s_{FC}^* is the selected global sign for the fully-connected layer.

The selected sign yields high prediction fidelity on the evaluated test samples, while the opposite sign yields a substantially lower value. Appendix C-D gives the procedures in Algorithms 6/7/8 for the input, hidden, and output layers, respectively.

F. Layer-Peeling Propagation

Multi-layer CNN extraction proceeds through sequential layer-by-layer parameter recovery. Each hidden layer is extracted by interpreting queried inputs through the previously recovered prefix. Figure 5 illustrates the layer-peeling workflow. The extraction proceeds from layer 0 to the FC layer. Each layer undergoes signature recovery followed by sign recovery before the next layer can be extracted. Scale factors accumulate through the network and are absorbed by the final FC layer.

1) Layer Peeling Strategy: The extraction attack processes layers sequentially from input to output. Layer 0 operates directly on the input space using the original hard-label oracle $\mathcal{O}_f$. After recovering layer i parameters $(\tilde{A}^{(i)}, \tilde{b}^{(i)})$, the algorithm evaluates later-layer constraints through the recovered prefix. The recovered prefix representation is defined by Eq. 34.

$$\tilde{h}^{(i)}(x) = \tilde{f}_i \circ \cdots \circ \tilde{f}_0(x) \tag{34}$$

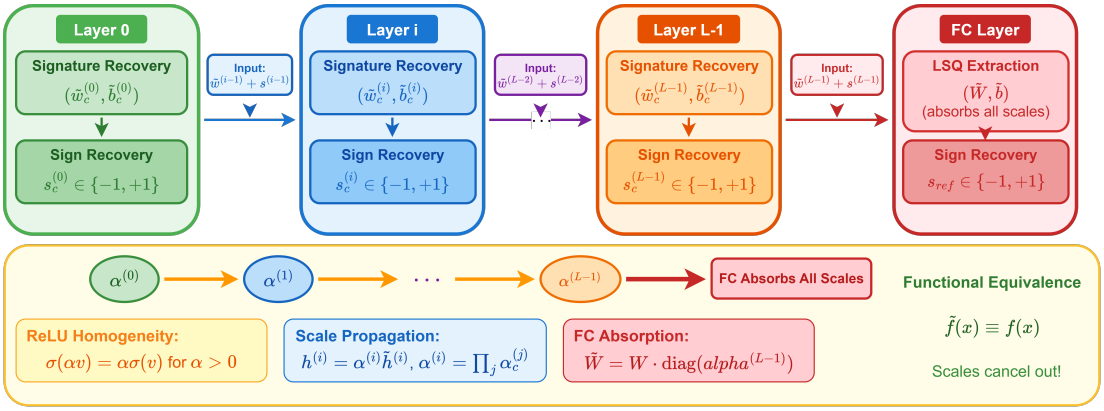

Fig. 5. Layer peeling with scale propagation. ReLU scale factors are absorbed by the FC layer.

Here $\tilde{f}_j$ denotes the reconstructed function for layer j using recovered parameters.

The layer peeling procedure applies signature recovery and sign recovery using original oracle labels and recovered prefix representations. For layer 0, dual points are interpreted directly in input space. For hidden layers $i > 0$, the same queried inputs are mapped to $\tilde{h}^{(i-1)}(x)$ before forming layer-local constraints. The FC layer extraction uses all recovered convolutional parameters to compute pooled feature representations. The total query count is the sum of the per-layer extraction costs. Algorithm 1 in Appendix C-A presents the complete procedure.

2) Scale Factor Propagation: SVD-based signature recovery produces unit-norm weight vectors. The true weights have unknown norms, introducing scale factors $\alpha_c^{(i)} \in \mathbb{R}_{>0}$ for each channel. The relationship between true and recovered weights is defined by Eq. 35.

$$w_c^{(i)} = \alpha_c^{(i)} \cdot \tilde{w}_c^{(i)} \tag{35}$$

Here $\tilde{w}_c^{(i)}$ is the recovered weight with $\|\tilde{w}_c^{(i)}\|_2 = 1$ and $\alpha_c^{(i)}$ is the unknown scale factor.

ReLU homogeneity $\sigma(\alpha v) = \alpha \cdot \sigma(v)$ allows scales to propagate through the network. Scale factors accumulate multiplicatively across layers without affecting activation patterns. The FC layer absorbs all accumulated scale factors through its weight matrix. This absorption preserves the hard-label decision boundaries under the scale ambiguities modeled here. Detailed analysis of scale propagation and bias error accumulation appears in Appendix B-C.

G. Complexity Analysis

The extraction cost decomposes into oracle query complexity and local computation for parameter recovery. Oracle queries are dominated by dual-point discovery and sign voting, while local computation is dominated by SVD, retained-candidate verification, and FC least squares. Let N denote the total number of oracle queries, $C_{\max}$ the maximum number of output channels in any layer, and p the per-layer flattened filter dimension.

1) Dual point finding query complexity: For each layer i, the algorithm allocates a path budget m_i over recoverable spatial neuron or class-pair units. For convolutional layers, these units include channel and spatial multiplicity, matching Table III. For example, the 3,136 Layer 0 paths equal the channel-position units counted by m_i, not an additional factor outside m_i. Each binary search requires $O(\log(1/\varepsilon))$ oracle queries for convergence threshold ε. Boundary walking requires $O(\kappa)$ queries per direction test, where κ is the subspace dimension. Let $m_{\max} = \max_i m_i$ denote the maximum per-layer path budget. The dual-point collection complexity is defined by Eq. 36.

$$T_{\text{dual}} = O(L \cdot m_{\max} \cdot \kappa \cdot \log(1/\varepsilon)) \tag{36}$$

Here T_{dual} denotes the oracle query count for dual-point collection, and $m_{\max}$ already includes the spatial paths induced by convolutional weight sharing. This term dominates extraction query complexity in the evaluated models.

2) Signature recovery computation complexity: For each layer, SVD is performed on an $m \times p$ dual point matrix per channel, where m is the number of dual points and $p = C_{\text{in}} \cdot k^2$ is the filter dimension. The SVD complexity is $O(\min(m,p)^2 \max(m,p))$ for each channel. When $m \approx p$ and the constraint system is well-conditioned, the per-channel cost is $O(p^3)$. The signature recovery time complexity is defined by Eq. 37.

$$T_{\text{signature}} = O(L_{\text{conv}} \cdot C_{\max} \cdot p^3) = O(L_{\text{conv}} \cdot C_{\max} \cdot (C_{\text{in}} \cdot k^2)^3). \tag{37}$$

Here L_{conv} is the number of convolutional layers, $C_{\max}$ is the maximum number of channels, and p is the filter dimension.

3) Sign recovery complexity: The sign recovery stage has two steps. The voting step probes each channel across pooled positions and validation points, while the verification step evaluates only the configurations induced by the low-confidence set. Let m_{vote} denote the number of validation probes per position, let B_i denote the number of retained sign candidates for layer i after confidence filtering, and let V_i denote the cost

of evaluating one retained candidate on the next layer. Eq. 38 defines the sign recovery cost.

$$T_{\text{sign}} = O\left(\sum_{i=0}^{L_{\text{conv}}-1} C_{\text{out}}^{(i)} \cdot n_{\text{patches}}^{(i)} \cdot m_{\text{vote}} + \sum_{i=0}^{L_{\text{conv}}-1} B_i \cdot V_i\right) \tag{38}$$

The first term is the channel-voting cost, and the second term is the verification cost for retained candidates. For non-final convolutional layers, V_i is the cost of one next-layer SVGR evaluation. For the final convolutional layer, V_i is the cost of one FC solvability check using the least-squares residual. In the evaluated models, $B_i \in \{2, 4, 8\}$, so sign recovery remains small compared with dual-point discovery.

4) FC layer extraction computation complexity: For the fully-connected output layer, a linear system of N_{FC} dual points solves for d weights per output neuron. Solving the least-squares problem requires $O(N_{\text{FC}} \cdot d^2)$ time. Typically $N_{\text{FC}} \approx d$ for an overdetermined system, giving $O(d^3)$ complexity per output neuron. The FC layer extraction complexity is defined by Eq. 39.

$$T_{\text{FC}} = O(K \cdot d^3). \tag{39}$$

Here K denotes the number of output classes and d denotes the weight dimension.

5) Overall complexity: The total complexity is polynomial when confidence filtering retains a polynomial number of sign candidates. The unfiltered full channel-sign space remains exponential in the number of unresolved channels. Eq. 40 defines the overall cost as the sum of query-driven and computation-driven stages.

$$T_{\text{total}} = T_{\text{dual}} + T_{\text{signature}} + T_{\text{sign}} + T_{\text{FC}}. \tag{40}$$

Each term represents one component of the extraction attack. With the retained-candidate bound, the dominant term is dual-point finding, which is polynomial in the number of recoverable path units and logarithmic in precision. For illustration, Eq. 41 estimates the query count when $L = 4$, $m_{\max} = 5{,}000$, and $\kappa = 10$.

$$N \approx 4 \cdot 5{,}000 \cdot 10 \cdot \log(10^6) \approx 2{,}760{,}000. \tag{41}$$

Here L is the number of extraction layers, $m_{\max}$ is the maximum path budget across layers, and κ is the subspace dimension.

6) Efficiency and defensive implications: The 1D MNIST experiment uses 2.37M oracle queries, and dual point discovery dominates the budget. This cost makes the attack query-costly despite polynomial asymptotic scaling. Table III reports this budget with the dual-point discovery statistics. Rate limiting, query auditing, randomized responses, output perturbation, and architecture hiding are practical countermeasures, but they do not prove intrinsic parameter secrecy for this model family.

V. Experiments

The evaluation applies the cryptanalytic extraction algorithm to CNNs trained on MNIST and CIFAR-10 under the stated hard-label threat model. The reported results cover dual-point discovery, signature recovery, sign recovery, and prediction fidelity. The evaluation answers four questions:

- Does dual-point discovery cover the recoverable channels?
- Are channel signatures and signs accurate enough for layer peeling?
- Does end-to-end recovery achieve prediction fidelity across the evaluated 1D, 2D, and RGB variants?
- What downstream security implications follow for watermark retention and wrapper backdoor composition?

The extraction metrics first establish that the recovered model matches the victim, while the security implication evaluation tests which watermark and wrapper behaviors remain after theft. The 1D MNIST target validates the flattened derivation, the 2D MNIST variants validate patch indexing and channel scaling, and the RGB CIFAR-10 variant validates multi-channel inputs.

A. Experimental Setup

The 1D MNIST target network is a 10-class classifier over 28×28 grayscale inputs flattened to 784 dimensions. It uses three Conv1D ReLU layers with 8, 8, and 4 channels, average pooling, and an FC output layer. The target models are controlled research CNNs for evaluating extraction fidelity rather than for training MNIST or CIFAR-10 classifiers. All experiments were executed on a workstation with an NVIDIA GeForce RTX 4090 GPU. The 2D and RGB target variants use the same extraction procedure, changing only patch and pooling indices while preserving the same hard-label oracle, ReLU, and average-pooling assumptions.

B. Dual Point Discovery Statistics

Table III reports the query budget for the extraction procedure in Section IV-C. Paths denote spatial neuron or class-pair units searched during dual-point discovery, clusters denote recovered dual-point groups, and class-pair units denote FC decision-boundary systems. Dual point discovery accounts for 99.6% of oracle queries, so boundary localization is the practical cost driver behind the complexity discussion in Section IV-G6.

Table IV reports dual-point discovery statistics for the procedure in Section IV-C.

Across the three convolutional layers, mean points per cluster range from 36.0 to 50.1, providing multiple constraints for signature recovery. Here clusters denote spatial neuron or path units, and signature recovery later aggregates same channel clusters for shared kernel SVD. For the FC layer, class-pair boundaries provide an overdetermined system with 15,004 constraints for least-squares recovery.

TABLE III
QUERY BUDGET BY EXTRACTION PHASE. DUAL POINT DISCOVERY DOMINATES TOTAL QUERIES.

Phase	Queries	Percentage
Dual Point Discovery	**2,363,020**	**99.6%**
Layer 0 (3,136 paths)	1,442,560	60.8%
Layer 1 (1,568 paths)	721,280	30.4%
Layer 2 (392 paths)	180,320	7.6%
FC Layer (41 class-pair units)	18,860	0.8%
Sign Recovery	**8,000**	**0.3%**
Voting (all layers)	$\approx$ 5,096	N/A
Discrete optimization	$\approx$ 2,904	N/A
Total	**2,371,020**	**100%**

TABLE IV
DUAL POINT COVERAGE BY LAYER. COUNTS REPORT COLLECTED POINTS AND POINTS PER CLUSTER.

Layer	Clusters	Dual Points	Channels	Points/Cluster
Layer 0 (Conv)	3,136	112,895	8	36.0
Layer 1 (Conv)	1,568	78,472	8	50.1
Layer 2 (Conv)	392	15,982	4	40.8
FC Layer	41 class-pair units	15,004	10	365.9
Total	**5,096**	**222,353**	**30**	**43.6**

TABLE V
SIGNATURE RECOVERY BY LAYER. METRICS REPORT COSINE SIMILARITY, RELATIVE ERROR, AND SVGR.

Layer	Raw CS	Adj CS	Rel Err	SVGR
Layer 0 (Conv)	1.0000	1.0000	2.03×10^{-8}	3.79×10^{-7}
Layer 1 (Conv)	0.9109	1.0000	4.12×10^{-7}	2.85×10^{-5}
Layer 2 (Conv)	0.9143	1.0000	1.98×10^{-7}	3.63×10^{-6}
FC Layer	0.9424	1.0000	1.47×10^{-6}	N/A
Mean	0.9419	1.0000	5.45×10^{-7}	N/A

C. Signature Recovery Analysis

Signature recovery accuracy is evaluated across all network layers. Table V summarizes the results. Formal metric definitions are provided in Appendix D-A.

Layer 0 achieves raw Cos-Sim 1.0000 and relative error 2.03×10^{-8} before sign adjustment, as reported in Table V. Layers 1 and 2 have lower raw Cos-Sim due to sign ambiguity, but adjusted Cos-Sim reaches 1.0000 for all channels after sign recovery with small errors. The FC layer also reaches adjusted Cos-Sim 1.0000 with slightly higher error from least squares. SVGR values indicate clear null-space separation across layers. Appendix D-B reports per-channel statistics.

D. Sign Recovery Analysis

Table VI summarizes two-step sign recovery, with method details in Section IV-E.

Table VI indicates that voting alone recovers 18/21 signs (85.7%) and leaves only a small low-confidence set in each convolutional layer. The follow-up optimization therefore evaluates only 8, 2, and 4 sign configurations for Layers 0, 1, and 2, respectively, rather than the full channel-sign spaces. After

TABLE VI
SIGN RECOVERY BEFORE AND AFTER OPTIMIZATION. LOW CONFIDENCE CHANNELS SET THE SEARCH SIZE.

Layer	Ch	Pre	Post	Low-Conf	#Comb
Layer 0 (Conv)	8	6/8 (75.0%)	8/8 (100%)	3 [1,3,4]	8
Layer 1 (Conv)	8	7/8 (87.5%)	8/8 (100%)	1 [3]	2
Layer 2 (Conv)	4	4/4 (100%)	4/4 (100%)	2 [0,2]	4
FC Layer	1 (global)	1/1 (100%)	1/1 (100%)	0	2
Total	**21**	**18/21 (85.7%)**	**21/21 (100%)**	N/A	N/A

SVGR-guided optimization, all convolutional signs and the FC global sign are recovered, giving 21/21 final accuracy.

The verification signal is decisive in the ambiguous cases. For Layer 0, the selected configuration achieves SVGR 5.728×10^{-7}, while competing configurations range from 4.163×10^{-1} to 6.943×10^{-1}. The same next-layer SVGR rule separates ambiguous signs in Layer 1, while Layer 2 uses FC solvability because it is the final convolutional layer. For the FC layer, the two possible global signs are separated directly by prediction fidelity on 1,000 test samples: $+1$ gives 0/1000 matches, while -1 gives 1000/1000 matches. Detailed per-channel voting and discrete optimization diagnostics are provided in Appendix D-B.

E. End-to-End Prediction-Fidelity Evaluation

Prediction fidelity (PF) measures sampled functional agreement on MNIST train, test, and random inputs, with definitions in Appendix D-A. Table VII compares the extracted model to distillation baselines across accuracy and PF metrics.

TABLE VII
PREDICTION FIDELITY AGAINST DISTILLATION. RESULTS COMPARE ACCURACY AND AGREEMENT.

Method	Queries	Train Acc	Test Acc	Train PF	Test PF	Random PF
Original	N/A	87.07%	87.52%	N/A	N/A	N/A
Distill-Test	1K	16.49%	16.69%	16.69%	16.95%	1.62%
Distill-Test	10K	44.76%	45.54%	47.14%	48.01%	0.72%
Distill-Random	10K	9.88%	10.09%	10.31%	10.24%	48.89%
Distill-Random	100K	9.93%	10.11%	10.36%	10.32%	81.08%
Distill-Random	1M	44.76%	44.31%	46.51%	45.52%	90.65%
Distill-Random	2M	22.76%	21.75%	23.25%	22.51%	90.27%
Distill-Random	10M	46.02%	46.02%	48.02%	47.84%	97.21%
Ours	2,371,020	**87.07%**	**87.52%**	**100.00%**	**100.00%**	**100.00%**

The extraction attack achieves 100.00% prediction fidelity on the train, test, and random-input splits, while distillation baselines remain far lower even with more queries. Distillation is included only as a predictive-mimicry baseline rather than as a direct cryptanalytic extraction baseline, since it targets task accuracy instead of exact agreement on random or boundary-probe inputs. Further details appear in Appendix D-E, Table III, and Appendix D-D.

F. Generalization and Scalability

Table VIII reports experiments across 1D, 2D, and RGB CNN variants.

These experiments vary input dimensionality, channel count, convolution type, and model width while preserving the structural assumptions of ReLU activations, average pooling, and

TABLE VIII
CNN SCALABILITY ACROSS EVALUATED VARIANTS. PF DENOTES TEST PREDICTION FIDELITY.

Dataset	Input	Conv	Architecture	Neurons	Rel. Err.	Test PF	Queries
MNIST [16]	784×1	1D	Conv(8-8-4)-FC(10)	5,096	5.5×10^{-7}	100.00%	2.37M
MNIST [16]	$28 \times 28 \times 1$	2D	Conv(4-4-4)-FC(10)	846	7.5×10^{-8}	100.00%	0.39M
MNIST [16]	$28 \times 28 \times 1$	2D	Conv(8-8-4)-FC(10)	1,666	4.6×10^{-7}	100.00%	0.76M
MNIST [16]	$28 \times 28 \times 1$	2D	Conv(16-16-4)-FC(10)	3,306	1.3×10^{-6}	100.00%	1.54M
CIFAR-10 [30]	$32 \times 32 \times 3$	2D	LeNet(6-16-120)-FC(10)	8,662	4.2×10^{-5}	100.00%	≈4.03M

architecture knowledge. Moving from 1D to 2D changes patch indexing and channel dimensions, but it does not change the dual-point, channel-signature, or layer-peeling principles. The CIFAR-10 experiment uses RGB inputs and a LeNet-style architecture with a 120 channel final convolutional layer. Prediction fidelity (PF) is measured on the test set, and query counts report the full extraction attack.

The results indicate that multiple 1D, 2D, and RGB CNN variants reach 100% test prediction fidelity. Query cost increases with the number of recovered neurons and channels, and the RGB LeNet-style setting is the largest tested configuration. The CIFAR-10 row reports an approximate query budget of 4.03M, obtained by scaling the measured per unit query rate from Table III to 8,662 recoverable units. These 2D and RGB results should be read as variants of the same extraction setting rather than as separate theoretical settings.

G. Security Implications

Parameter recovery changes provenance and abuse analysis because extracted models may preserve security relevant behavior. The evaluation applies the same extraction attack to five watermarked CNNs covering trigger, parameter, and feature or activation mechanisms [31]–[34]. It reports prediction fidelity, behavioral watermark retention, and parameter bit retention separately because functionally equivalent extraction can preserve behavior without preserving every bit-level encoding.

Table IX compares extraction with hard-label distillation and soft KD distillation using 10M queries [2], [35], [36]. WA is watermark advantage, WSR is raw watermark success rate, and HL, KD, and Ext. denote hard-label distillation, soft KD distillation, and the extracted model.

TABLE IX
WATERMARK RETENTION AFTER EXTRACTION. VALUES REPORT WATERMARK METRICS AND BIT ACCURACY.

WM	Metric	Orig.	HL	KD	Ext.
Patch	WA(WSR)	91.50 (93.25)	2.00 (27.00)	25.25 (49.25)	91.50 (93.25)
Blended	WA(WSR)	97.00 (100.00)	0.00 (100.00)	0.00 (100.00)	97.00 (100.00)
Content/mask	WA(WSR)	94.00 (100.00)	0.00 (100.00)	1.33 (100.00)	94.00 (100.00)
Parameter	Bit Acc.	93.75	53.12	65.62	79.69
Feature/act.	WA(WSR)	92.00 (96.33)	0.67 (100.00)	32.33 (100.00)	92.00 (96.33)
Feature/act.	Bits	78.10	84.57	78.96	78.10

Table X reports whether ownership evidence survives extraction while a deterministic wrapper adds trigger behavior. BD-ASR is measured over four wrapper triggers per watermark model, and FH denotes false hits on 10,000 clean inputs.

Recent NDSS work treats model theft, ownership verification, and trigger behavior as connected deployment risks.

TABLE X
WRAPPER BACKDOOR RETENTION. RESULTS REPORT TRIGGER SUCCESS, FALSE HITS, AND WATERMARK BEHAVIOR.

WM	Metric	Orig.	Ext.	FH	WM after
Patch	BD-ASR/WA(WSR)	100.00	100.00	0/10k	92.50 (94.25)
Blended	BD-ASR/WA(WSR)	100.00	100.00	0/10k	97.00 (100.00)
Content/mask	BD-ASR/WA(WSR)	100.00	100.00	0/10k	94.00 (100.00)
Parameter	BD-ASR/Bit Acc.	100.00	100.00	0/10k	79.69
Feature/act.	BD-ASR/WA/Bits	100.00	100.00	0/10k	92.00 (96.33)/78.10

SSLExtraction studies model extraction against watermarked APIs and watermark removal through query selection [37]. EaaW frames ownership verification as checking whether suspicious models retain owner selected verification behavior [38]. QuRA and CrowdGuard treat trigger controlled behavior as a deployed model risk across quantization and federated training [39], [40]. Our experiment places hard-label cryptanalytic extraction at this intersection. The stolen model preserves behavior-level watermark evidence more reliably than distillation, while a wrapper adds trigger behavior without erasing the provenance signal. For model owners, the result means watermark evidence can remain useful after theft, but parameter equality checks alone may miss derivative services. For defenders and downstream users, the recovered service can appear provenance linked and clean on ordinary inputs while still exposing trigger controlled behavior. This composition makes auditing after extraction necessary.

VI. DISCUSSION AND LIMITATIONS

The attack applies to deterministic known-architecture CNN classifiers with ReLU activations, average pooling, and classification outputs. The formal derivation uses the 1D formulation, while the 2D and RGB variants use the same extraction logic with different patch and pooling indices. Unknown architectures, max pooling, smooth activations, residual networks, regression, segmentation, and other structured-output tasks remain outside scope. Average pooling is the focus because it is linear and absorbable into later linear targets. Existing max-pooling CNN extraction studies rely on raw-output or soft-label access, where output magnitudes reveal pooling-switching events [8], [9]. Hard-label access collapses these numeric signals to a single class index, so max-pooling extraction requires new observability conditions.

The query cost limits direct deployment as a low-cost MLaaS attack, and the 1D MNIST experiment uses 2.37M hard-label queries. Rate limiting, query auditing, randomized responses, output perturbation, anomaly detection, and architecture hiding can raise the practical cost. The watermark and wrapper experiments demonstrate a broader provenance risk after extraction, because the recovered model can preserve ownership evidence while acquiring deterministic trigger behavior. This composition result motivates provenance checks that go beyond parameter equality and clean prediction fidelity. The evaluation uses only public datasets and locally trained

models, and the code implementation is available for reproducibility.

VII. Conclusion

This paper presents an end-to-end hard-label cryptanalytic extraction attack for known-architecture CNN classifiers with ReLU activations and average pooling. The contribution is complementary to prior raw-output and soft-label max-pooling extraction, and it addresses the hard-label setting where numeric pooling-switch signals are unavailable. The main method innovation is channel-level recovery with SVGR-guided retained-candidate discrete optimization under the stated assumption. Across evaluated 1D, 2D, and RGB variants, the extracted models reach 100% prediction fidelity under hard-label access. The results demonstrate that hiding logits alone is not a sufficient parameter-secrecy boundary for this model family once architecture information is available.

Generative AI Usage. OpenAI ChatGPT was used for grammar and clarity edits across parts of the manuscript. All technical content, experimental results, and citations were produced and verified by the authors.

References

[1] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 36–52.

[2] J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, "Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[3] D. Oliynyk, R. Mayer, and A. Rauber, "I know what you trained last summer: A survey on stealing machine learning models and defences," *ACM Comput. Surv.*, vol. 55, no. 14s, Jul. 2023.

[4] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 601–618. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer

[5] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020.

[6] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 189–218.

[7] Y. Chen, X. Dong, J. Guo, Y. Shen, A. Wang, and X. Wang, "Hard-label cryptanalytic extraction of neural network models," in *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*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 207–236. [Online]. Available: https://doi.org/10.1007/978-981-96-0944-4_7

[8] Z. Chen, S. Tang, Z. Gao, Y. Su, L. Qin, and X. Dong, "Algebraic attack on convolutional neural network with max pooling," IACR Cryptology ePrint Archive, Paper 2026/241, 2026. [Online]. Available: https://eprint.iacr.org/2026/241

[9] H. Liu, A. Siproudhis, C. Boura, and T. Peyrin, "Model extraction of convolutional neural networks with max-pooling," IACR Cryptology ePrint Archive, Paper 2026/464, 2026. [Online]. Available: https://eprint.iacr.org/2026/464

[10] I. A. Canales-Martínez, J. Chávez-Saab, A. Hambitzer, F. Rodríguez-Henríquez, N. Satpute, and A. Shamir, "Polynomial time cryptanalytic extraction of neural network models," in *Advances in Cryptology – EUROCRYPT 2024*, M. Joye and G. Leander, Eds. Cham: Springer Nature Switzerland, 2024, pp. 3–33.

[11] N. Carlini, J. Chávez-Saab, A. Hambitzer, F. Rodríguez-Henríquez, and A. Shamir, "Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting," in *Advances in Cryptology – EUROCRYPT 2025*, S. Fehr and P.-A. Fouque, Eds. Cham: Springer Nature Switzerland, 2025, pp. 364–396.

[12] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "DeepSteal: Advanced model extractions leveraging efficient weight stealing in memories," in *Proceedings of the 2022 IEEE Symposium on Security and Privacy*, 2022, pp. 1157–1174.

[13] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal, "DeepTheft: Stealing dnn model architectures through power side channel," in *Proceedings of the 2024 IEEE Symposium on Security and Privacy*, 2024, pp. 3311–3326.

[14] I. A. Canales-Martínez and D. Santos, "Extracting some layers of deep neural networks in the hard-label setting," Cryptology ePrint Archive, Paper 2025/1118, 2025. [Online]. Available: https://eprint.iacr.org/2025/1118

[15] X. Sun, H. Lei, L. Wei, X. Qi, K. Hu, M. Wang, and W. Wang, "Cryptanalytic extraction of convolutional neural networks," *IACR Cryptol. ePrint Arch.*, vol. 2026, p. 139, 2026. [Online]. Available: https://eprint.iacr.org/2026/139

[16] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist

[17] D. Rolnick and K. P. Körding, "Reverse-engineering deep relu networks," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20. JMLR.org, 2020.

[18] H. Foerster, R. Mullins, I. Shumailov, and J. Hayes, "Beyond slow signs in high-fidelity model extraction," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 19 496–19 522.

[19] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring connections between active learning and model extraction," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020.

[20] K. Krishna, G. S. Tomar, A. P. Parikh, N. Papernot, and M. Iyyer, "Thieves on sesame street! model extraction of bert-based apis," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=Byl5NREFDr

[21] S. Addepalli, G. K. Nayak, A. Chakraborty, and V. B. Radhakrishnan, "Degan: Data-enriching gan for retrieving representative samples from a trained classifier," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3130–3137.

[22] M. Biton Dor and Y. Mirsky, "Efficient model extraction via boundary sampling," in *Proceedings of the 2024 Workshop on Artificial Intelligence and Security*, ser. AISec '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–11. [Online]. Available: https://doi.org/10.1145/3689932.3694756

[23] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, no. 3. PMLR, 2013, pp. 1139–1147. [Online]. Available: https://proceedings.mlr.press/v28/sutskever13.html

[24] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. A. Smith, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," *CoRR*, vol. abs/2002.06305, 2020. [Online]. Available: https://arxiv.org/abs/2002.06305

[25] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of DNNs," in *Advances in Neural Information Processing Systems*, vol. 31, 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/be3087e74e9100d4bc4c6268cdbe8456-Abstract.html

[26] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 3519–3529. [Online]. Available: https://proceedings.mlr.press/v97/kornblith19a.html

[27] J. Mehrer, C. J. Spoerer, N. Kriegeskorte, and T. C. Kietzmann, "Individual differences among deep neural network models," *Nature Communications*, vol. 11, no. 1, p. 5725, 2020. [Online]. Available: https://www.nature.com/articles/s41467-020-19632-w

[28] C. Fefferman and S. Markel, "Recovering a feed-forward net from its output," in *Proceedings of the 7th International Conference on Neural Information Processing Systems*, ser. NIPS'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 335–342.

[29] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Csi nn: reverse engineering of neural network architectures through electromagnetic side channel," in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19. USA: USENIX Association, 2019, pp. 515–532.

[30] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744

[31] Y. Adi, C. Baum, M. Cissé, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, 2018, pp. 1615–1631.

[32] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. Association for Computing Machinery, 2018, pp. 159–172.

[33] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. Association for Computing Machinery, 2017, pp. 269–277.

[34] B. D. Rouhani, H. Chen, and F. Koushanfar, "DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. Association for Computing Machinery, 2019, pp. 485–497.

[35] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4954–4963.

[36] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[37] H. Luan, X. Tan, Z. Li, J. Dai, X. Sun, and P. Chen, "Dataset reduction and watermark removal via self-supervised learning for model extraction attack," in *Proceedings 2026 Network and Distributed System Security Symposium*. Internet Society, 2026.

[38] S. Shao, Y. Li, H. Yao, Y. He, Z. Qin, and K. Ren, "Explanation as a watermark: Towards harmless and multi-bit model ownership verification via watermarking feature attribution," in *Proceedings 2025 Network and Distributed System Security Symposium*. Internet Society, 2025.

[39] X. Chen, P. Zhang, J. Sun, W. Wang, and J. Wang, "Rounding-guided backdoor injection in deep learning model quantization," in *Proceedings 2026 Network and Distributed System Security Symposium*. Internet Society, 2026.

[40] P. Rieger, T. Krauss, M. Miettinen, A. Dmitrienko, and A.-R. Sadeghi, "Crowdguard: Federated backdoor detection in federated learning," in *Proceedings 2024 Network and Distributed System Security Symposium*. Internet Society, 2024.

APPENDIX A
NOTATION SUMMARY

This appendix gives a complete reference of mathematical notation organized by category. Table XI summarizes model and parameter notation. Table XII presents geometry and algorithm notation. Table XIII details recovery and evaluation notation.

TABLE XI
MODEL AND PARAMETER NOTATION. SYMBOLS COVER LAYERS, CHANNELS, KERNELS, AND RECOVERED PARAMETERS.

Symbol	Description	Domain
Network Architecture		
f, $\tilde{f}$	Target / extracted logit network	$\mathbb{R}^{d_0} \to \mathbb{R}^K$
f_i, $\tilde{f}_i$	Layer i function / reconstructed layer	$\mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$
L, L_{conv}, i	Total extraction layers / convolutional layers / layer index	$\mathbb{N}^+$, $\mathbb{N}^+$, $\{0, \dots, L-1\}$
d_i, d_0, K	Layer i dim / input dim / classes	$\mathbb{N}^+$
$\sigma(\cdot)$	ReLU activation function	$\mathbb{R} \to \mathbb{R}_{\geq 0}$
Convolutional Layer Parameters		
$C_{\text{in}}^{(i)}$, $C_{\text{out}}^{(i)}$	Input / output channels at layer i	$\mathbb{N}^+$
$k^{(i)}$, s	Kernel size / stride	$\mathbb{N}^+$
$n_{\text{patches}}^{(i)}$	Spatial positions at layer i	$\mathbb{N}^+$
$G_i(\cdot)$	Patch extraction operator (im2col)	$\mathbb{R}^{d_{i-1}} \to \mathbb{R}^{(C_{\text{in}} \cdot k) \times n_{\text{patches}}}$
$A^{(i)}$, $b^{(i)}$	Weight matrix / bias at layer i	$\mathbb{R}^{C_{\text{out}} \times (C_{\text{in}} \cdot k)}$, $\mathbb{R}^{C_{\text{out}}}$
W_{FC}, b_{FC}	FC layer weight / bias	$\mathbb{R}^{K \times d_{\text{pool}}}$, $\mathbb{R}^K$
W_j, b_j	FC weight / bias for class j	$\mathbb{R}^{d_{\text{pool}}}$, $\mathbb{R}$
Extracted Parameters		
$\bar{A}^{(i)}$, $\bar{b}^{(i)}$	Extracted weight / bias at layer i	$\mathbb{R}^{C_{\text{out}} \times (C_{\text{in}} \cdot k)}$, $\mathbb{R}^{C_{\text{out}}}$
$\bar{W}_{\text{FC}}$, $\bar{b}_{\text{FC}}$	Extracted FC weight / bias	$\mathbb{R}^{K \times d_{\text{pool}}}$, $\mathbb{R}^K$
$\hat{w}_c$, $\hat{w}_c$	Raw / normalized weight signature	$\mathbb{R}^{C_{\text{in}} \cdot k}$, unit sphere
$\bar{b}_c$	Bias estimated after centered SVD by median aggregation	$\mathbb{R}$
Indexing		
c, p	Channel / spatial position index	$\{1, \dots, C_{\text{out}}\}$, $\{1, \dots, n_{\text{patches}}\}$
η	Neuron identifier (c, p)	Channel $\times$ Position
w_c, b_c, w_η, b_η	Weight / bias for channel c or neuron η	$\mathbb{R}^{C_{\text{in}} \cdot k}$, $\mathbb{R}$
j, ℓ, ℓ_{ref}	Class indices / reference class	$\{0, \dots, K-1\}$

TABLE XII
GEOMETRY AND ALGORITHM NOTATION. SYMBOLS DEFINE DUAL POINTS, SUBSPACES, AND SVD TERMS.

Symbol	Description	Domain
Layer Activations		
$h^{(i)}$, $\tilde{h}^{(i)}$	True / propagated activation at layer i	$\mathbb{R}^{C_{\text{out}}^{(i)} \times n_{\text{patches}}^{(i)}}$
$h_{c,p}^{(i)}$	Activation at channel c, position p	$\mathbb{R}$
$\bar{h}$, $\bar{h}_c$	Pooled feature vector / for channel c	$\mathbb{R}^{d_{\text{pool}}}$, $\mathbb{R}$
Geometric Objects		
$\mathcal{H}_\eta$	Critical hyperplane of neuron η	Subset of $\mathbb{R}^{d_0}$
$\mathcal{B}_{j,\ell}$	Decision boundary between classes j, ℓ	Subset of $\mathbb{R}^{d_0}$
$g(x)$	Gap function: $f(x)[j_1] - f(x)[j_2]$	$\mathbb{R}$
$z(\eta, x)$	Pre-activation value of neuron η at x	$\mathbb{R}$
n_b, t_b	Boundary normal / tangent direction	$\mathbb{R}^{d_0}$
Dual Point Finding		
x^*, x_b, x_{dual}	Dual point / boundary point / discovered dual	$\mathbb{R}^{d_0}$
x_{mid}, x_{end}	Context points around dual point	$\mathbb{R}^{d_0}$
(x^+, x^-)	Dual point pair	$\mathbb{R}^{d_0} \times \mathbb{R}^{d_0}$
τ, τ^*, τ_{step}	Step size / max step / exploration step	$\mathbb{R}^+$
Subspace and Boundary Walking		
$\mathcal{U}$	Low-dimensional subspace	Subset of $\mathbb{R}^{d_0}$
$\{v_1, \dots, v_\kappa\}$, κ	Orthonormal basis / subspace dimension	$\mathbb{R}^{d_0}$, $\mathbb{N}^+$
δ_j^+, δ_j^-	Forward / backward boundary distances	$\mathbb{R}^+$
ε, θ, r	Precision / angle / random direction	$\mathbb{R}^+$, $[0, \pi/2]$, $\mathbb{R}^{d_0}$
Signature Extraction		
D_c	Centered dual point constraint matrix for channel c	$\mathbb{R}^{N_c \times (C_{\text{in}} \cdot k)}$
N_c, m	Dual points for channel c / per channel	$\mathbb{N}^+$
U, Σ, V	SVD decomposition matrices	N/A
ς_j, $\varsigma_{\min}$, $\varsigma_{\min+1}$	Singular values (from SVD)	$\mathbb{R}^+$

Conventions. Layer index i, channel index c, position index p. Neuron $\eta = (c, p)$. Class indices j, ℓ. ReLU is denoted $\sigma(\cdot)$, singular values ς_j are distinct from σ, and subspace dimension κ is distinct from kernel k.

TABLE XIII
RECOVERY AND EVALUATION NOTATION. SYMBOLS DEFINE SIGNS, ACCURACY, FIDELITY, AND QUERIES.

Symbol	Description	Domain
Sign Recovery		
s_c, s_{FC}	Recovered sign for channel c / FC layer	$\{-1, +1\}$
$\mathcal{S}$, $\mathbf{s}$, $\mathbf{s}^*$	Sign space / configuration / optimal	$\{-1, +1\}^{C_{\text{out}}}$
$v_{c,p}$	Input-space perturbation direction	$\mathbb{R}^{d_0}$
$\Delta(\cdot)$	Response indicator function	$\{0, 1\}$
Δ_{on}, Δ_{off}	On-side / off-side boundary distances	$\mathbb{R}^+$
$\nu_{c,p}$, n_+, n_-	Position vote / positive / negative counts	$\mathbb{Z}$, $\mathbb{N}$
conf_c	Confidence score for channel c	$[0, 1]$
$\mathcal{C}_{\text{high}}$, $\mathcal{C}_{\text{low}}$	High / low confidence channel sets	N/A
$\mathcal{J}^{(i)}$	Jacobian of layers 0 through $i-1$	$\mathbb{R}^{d_i \times d_0}$
$\text{unfold}(\cdot, p)$	Unfold kernel weights to input space	$\mathbb{R}^{C_{\text{in}} \cdot k} \to \mathbb{R}^{d_0}$
Scale Propagation and Error		
$\alpha_c^{(i)}$, $\alpha^{(i)}$	Scale factor for channel c / vector	$\mathbb{R}^+$, $\mathbb{R}_{>0}^{C_{\text{out}}^{(i)}}$
$\tilde{h}^{(i)}(x)$	Recovered prefix representation after layer i	$\mathbb{R}^{d_i}$
$A_{\text{rec}}^{(i)}$, $b_{\text{rec}}^{(i)}$	Recovered weight / bias (unit-norm)	$\mathbb{R}^{C_{\text{out}} \times (C_{\text{in}} \cdot k)}$, $\mathbb{R}^{C_{\text{out}}}$
$\epsilon_c^{(i)}$, $\epsilon^{(i)}$	Numerical error for channel c / vector	$\mathbb{R}$, $\mathbb{R}^{C_{\text{out}}^{(i)}}$
$\Delta_j^{(i)}$, $\Delta^{(L)}$	Accumulated / cumulative error	$\mathbb{R}$
Oracle and Evaluation		
$\mathcal{O}_f$, $\mathcal{O}_{\tilde{f}}$	Hard-label oracle induced by target / extracted network	$\mathbb{R}^{d_0} \to \{0, \dots, K-1\}$
Q	Query count	$\mathbb{N}$
$\mathcal{D}$, $\mathcal{D}_c$, $\mathcal{C}$	Dual point collection / constraints	Set
γ	Scaling constant for symmetry-equivalent parameters	$\mathbb{R}^+$
SVGR	$\varsigma_{\min}/\varsigma_{\min+1}$	$[0, 1]$
Cos-Sim, PF	Cosine similarity / prediction fidelity	$[-1, 1]$, $[0, 1]$

APPENDIX B
METHODOLOGICAL SUPPLEMENT

A. Boundary Direction Estimation Details

This appendix gives the derivation and computational procedure for estimating the decision boundary normal vector at a boundary point x_b.

1) Distance Measurement: For each basis direction v_j in a chosen coordinate system $\{v_1, \dots, v_\kappa\}$, the distance from x_b to the decision boundary is measured in both positive and negative directions. The forward boundary distance is defined by Eq. 42.

$$\delta_j^+ = \inf\{\tau > 0 : \mathcal{O}_f(x_b + \tau v_j) \neq \mathcal{O}_f(x_b)\} \tag{42}$$

Here δ_j^+ represents the smallest positive step τ along direction v_j that crosses the decision boundary. The backward boundary distance is defined by Eq. 43.

$$\delta_j^- = \inf\{\tau > 0 : \mathcal{O}_f(x_b - \tau v_j) \neq \mathcal{O}_f(x_b)\}. \tag{43}$$

Here δ_j^- represents the smallest positive step τ in the negative direction along v_j that crosses the decision boundary.

2) Directional Derivative Approximation: The ratio of these distances approximates the directional derivative of the gap function g along v_j. Eq. 44 gives the directional derivative approximation.

$$\frac{\partial g}{\partial v_j}(x_b) \approx \frac{\delta_j^- - \delta_j^+}{\delta_j^- + \delta_j^+}. \tag{44}$$

The formula follows from the symmetry properties of linear functions around the zero level set. When $\delta_j^- > \delta_j^+$, the boundary is closer in the positive v_j direction, indicating that the normal points primarily in the negative v_j direction.

3) Efficient Computation: The distances δ_j^+ and δ_j^- are computed by logarithmic search over candidate step sizes. The candidate scales in Eq. 45 are tested at points $x_b + sv_j$ to find an approximate crossing scale.

$$s \in \{10^{-7}, 10^{-6.5}, 10^{-6}, \ldots, 10^{0}\} \tag{45}$$

Let $s_{\min}$ be the smallest value where $\mathcal{O}_f(x_b + sv_j) \neq \mathcal{O}_f(x_b)$. Binary search between $s_{\min}/2$ and $s_{\min}$ then refines δ_j^+ to precision ε. A symmetric procedure yields δ_j^-.

4) Subspace Projection: To reduce query complexity from $O(d_0 \log(1/\varepsilon))$ to $O(\kappa \log(1/\varepsilon))$, normal estimation is restricted to a κ-dimensional subspace $\mathcal{U} \subset \mathbb{R}^{d_0}$ where $\kappa \ll d_0$. Choose an orthonormal basis $\{v_1, \ldots, v_\kappa\}$ for $\mathcal{U}$, typically by setting $v_1 = x_b/\|x_b\|$ and sampling $v_2, \ldots, v_\kappa$ from a standard Gaussian followed by Gram-Schmidt orthogonalization. The partial normal $n_b^{(\mathcal{U})}$ approximates the projection of the true normal onto $\mathcal{U}$. This approximation suffices for boundary walking because decision boundaries are locally low-dimensional near dual points.

B. Boundary Walking Details

This appendix gives the mathematical formulation for boundary walking to discover dual points.

1) Tangent Direction Computation: Select a random direction $r \in \mathbb{R}^{d_0}$ within the chosen subspace $\mathcal{U}$. The tangent projection is defined by Eq. 46.

$$t_b = r - \frac{r \cdot n_b}{n_b \cdot n_b} n_b. \tag{46}$$

Here t_b is the projection of r onto the tangent space by subtracting the normal component. The vector t_b satisfies $t_b \perp n_b$, ensuring it lies in the tangent space $T_{x_b}\mathcal{B}$. Normalize to obtain the unit tangent direction $\hat{t}_b = t_b/\|t_b\|_2$.

2) Boundary Proximity Testing: The walk candidate point is defined by Eq. 47.

$$x_{\text{cand}}(\tau) = x_b + \tau \hat{t}_b \tag{47}$$

Here $\tau > 0$ is the step size parameter. Neighborhood tests determine whether $x_{\text{cand}}(\tau)$ remains on the decision boundary. Sample random perturbations $\{\delta_j\}_{j=1}^{m_{\text{test}}}$ with $\|\delta_j\|_2 \leq \varepsilon_{\text{test}}$. If any perturbation satisfies $\mathcal{O}_f(x_{\text{cand}}(\tau) + \delta_j) \neq \mathcal{O}_f(x_{\text{cand}}(\tau) - \delta_j)$, then $x_{\text{cand}}(\tau)$ is treated as near the boundary.

3) Maximum Step Size Search: Perform binary search over τ to find the maximum step size τ^* where the boundary proximity test succeeds. The maximum step size is defined by Eq. 48.

$$\tau^* = \sup \Big\{ \tau \geq 0 : \exists\, \delta \text{ with } \|\delta\|_2 \leq \varepsilon_{\text{test}}, \\ \mathcal{O}_f(x_b + \tau \hat{t}_b + \delta) \neq \mathcal{O}_f(x_b + \tau \hat{t}_b - \delta) \Big\}. \tag{48}$$

Here τ^* represents the supremum of step sizes for which the boundary proximity test succeeds. When $\tau > \tau^*$, the point $x_b + \tau \hat{t}_b$ has departed from the boundary neighborhood, typically because a neuron has switched activation and changed the local decision boundary geometry.

4) Dual Point Localization: The step size τ^* indicates proximity to a dual point. Small values ($\tau^* < 10^{-4}$) suggest no nearby critical point, while large values ($\tau^* > 10^{-2}$) indicate a long linear segment. Intermediate values signal a bend in the boundary caused by a neuron activation change.

The midpoint and endpoint along the walk are defined by Eq. 49.

$$x_{\text{mid}} = x_b + \frac{\tau^*}{2} \hat{t}_b, \quad x_{\text{end}} = x_b + \tau^* \hat{t}_b. \tag{49}$$

Here x_{mid} serves as the left context point and x_{end} provides the right context for the dual point. To refine the dual point location, overstep slightly past x_{end} along $\hat{t}_b$ to ensure crossing the decision boundary. Apply binary search between x_{end} and $x_{\text{end}} + \delta_{\text{step}} \hat{t}_b$ (with $\delta_{\text{step}} = 10^{-4}$) to locate the precise boundary crossing x_{dual}.

C. Scale and Bias Propagation Analysis

Scale factors and bias errors propagate through the extracted network in different ways.

1) Scale Factor Propagation: The signature recovery algorithm produces unit-norm weight vectors through SVD normalization. The true network weights have arbitrary norms determined by training dynamics. This discrepancy introduces scale factors that propagate through the network architecture.

Let $w_c^{(i)} \in \mathbb{R}^{C_{\text{in}}^{(i)} \cdot k^{(i)}}$ denote the true weight vector for output channel c at layer i. Let $\tilde{w}_c^{(i)}$ denote the recovered unit-norm weight vector satisfying $\|\tilde{w}_c^{(i)}\|_2 = 1$. The relationship between true and recovered weights is defined by Eq. 50.

$$w_c^{(i)} = \alpha_c^{(i)} \cdot \tilde{w}_c^{(i)} \tag{50}$$

Here $\alpha_c^{(i)} \in \mathbb{R}_{>0}$ is the unknown scale factor for channel c at layer i.

The ReLU activation function exhibits a critical homogeneity property that enables scale factorization. For any vector $v \in \mathbb{R}^n$ and positive scalar $\alpha > 0$, the homogeneity property is defined by Eq. 51.

$$\sigma(\alpha \cdot v) = \alpha \cdot \sigma(v) \tag{51}$$

This property allows scale factors to propagate through ReLU layers by multiplicative accumulation.

Consider the forward propagation through two consecutive layers. Let $h^{(i)}$ denote the input to layer i and $h^{(i+1)}$ the output after activation. Using recovered parameters with scale factors, the forward pass is represented in Eq. 52.

$$h^{(i+1)} = \sigma\left(\text{diag}(\alpha^{(i)}) \cdot A_{\text{rec}}^{(i)} \cdot G_i(h^{(i)}) + b_{\text{rec}}^{(i)}\right) \tag{52}$$

Here $\text{diag}(\alpha^{(i)})$ is the diagonal matrix of channel-wise scale factors and $A_{\text{rec}}^{(i)}$ contains the recovered weights with $\|w_c^{(i)}\|_2 = 1$ for each channel c.

Applying the ReLU homogeneity property from Eq. 51 yields the scale factorization in Eq. 53.

$$h^{(i+1)} = \text{diag}(\alpha^{(i)}) \cdot \sigma\left(A_{\text{rec}}^{(i)} \cdot G_i(h^{(i)}) + \text{diag}(\alpha^{(i)})^{-1} \cdot b_{\text{rec}}^{(i)}\right) \tag{53}$$

This factorization reveals that scale factors affect subsequent layers through their input activations.

For an L-layer network, scale factors accumulate multiplicatively from input to output. The effective scale for channel c at layer i is defined by Eq. 54.

$$\alpha_{c,\text{eff}}^{(i)} = \prod_{j=0}^{i-1} \alpha_{c'}^{(j)} \tag{54}$$

Here c' denotes the channels that contribute to channel c through the convolution operation.

The multiplicative accumulation creates numerical sensitivity in deep networks. Small relative errors in early layer scales compound exponentially through subsequent layers. For a network with uniform scale error ϵ per layer, the cumulative error at layer L scales as $(1+\epsilon)^L$. This exponential growth limits extraction accuracy for deeper architectures without scale correction.

2) Bias Error Accumulation: While scale factors propagate multiplicatively through ReLU activations, bias terms introduce additive errors that accumulate across layers. The dual point analysis recovers bias values modulo the unknown scale factors. For channel c at layer i, the relationship between true bias $b_c^{(i)}$ and recovered bias $\tilde{b}_c^{(i)}$ is defined by Eq. 55.

$$\tilde{b}_c^{(i)} = \frac{b_c^{(i)}}{\alpha_c^{(i)}} + \epsilon_c^{(i)} \tag{55}$$

Here $\epsilon_c^{(i)}$ represents numerical errors from the median estimation process.

The forward propagation through recovered layers accumulates these bias errors. Consider the activation pattern for neuron j at layer $i+1$ with input $h^{(i+1)}$. The pre-activation value using recovered parameters is represented in Eq. 56.

$$\begin{aligned} &\langle \hat{w}_j^{(i+1)}, G_{i+1}(h^{(i+1)})[:,p] \rangle + \hat{b}_j^{(i+1)} \\ &= \frac{1}{\alpha_j^{(i+1)}} \Big(\langle w_j^{(i+1)}, \alpha^{(i)} \odot G_{i+1}(h^{(i+1)})[:,p] \rangle \\ &\quad + b_j^{(i+1)} \Big) + \Delta_j^{(i+1)} \end{aligned} \tag{56}$$

Here $\alpha^{(i)} \odot G_{i+1}(h^{(i+1)})[:,p]$ denotes channel-wise scaling and $\Delta_j^{(i+1)}$ is the accumulated error term.

The accumulated error grows with network depth through two mechanisms. First, bias errors from early layers affect the input distribution to later layers through the activation function. Second, scale factor uncertainty amplifies the impact of bias errors when propagated to subsequent layers. The cumulative error for layer L is defined by Eq. 57.

$$\Delta^{(L)} \leq \sum_{i=0}^{L-1} \left(\prod_{j=i+1}^{L-1} \|\alpha^{(j)}\|_\infty \right) \cdot \|\epsilon^{(i)}\|_\infty \tag{57}$$

The product accounts for error amplification through multiplicative scale propagation.

Despite this accumulation, the hard-label oracle returns only the predicted class label and therefore depends only on the ordering of the final logits. Classification decisions remain invariant to uniform scaling of all class logits. As long as the accumulated errors preserve the relative ordering of logit values, prediction fidelity is maintained on those inputs. For networks with sufficient margin separation between classes, moderate bias errors do not affect prediction accuracy.

3) FC Layer Scale Absorption: After recovering all convolutional layers, the FC layer extraction absorbs accumulated scale factors. Let $x^{(L-1)} \in \mathbb{R}^{d_{\text{pool}}}$ denote the feature vector before the FC layer after average pooling. The pooling operator computes fixed linear averages over the final feature map. For channel c with spatial resolution $n_{\text{patches}}^{(L-1)}$, the pooled feature is defined by Eq. 58.

$$\bar{x}_c^{(L-1)} = \frac{1}{n_{\text{patches}}^{(L-1)}} \sum_{p=1}^{n_{\text{patches}}^{(L-1)}} x_{c,p}^{(L-1)} \tag{58}$$

Here $x_{c,p}^{(L-1)}$ denotes the activation for channel c at spatial position p.

Average pooling commutes with positive scalar multiplication due to its linearity. The pooled features using recovered parameters are represented in Eq. 59.

$$\bar{x}_{\text{rec}}^{(L-1)} = \text{diag}(\alpha^{(L-1)})^{-1} \cdot \bar{x}_{\text{true}}^{(L-1)} \tag{59}$$

Here $\alpha^{(L-1)}$ contains the accumulated scale factors at the final convolutional layer.

The FC layer performs a linear transformation without activation for logit computation. Let $W_{\text{FC}} \in \mathbb{R}^{K \times d_{\text{pool}}}$ denote the true FC weight matrix with K classes. The true logits for input x are defined by Eq. 60.

$$\ell_{\text{true}} = W_{\text{FC}} \cdot \bar{x}_{\text{true}}^{(L-1)} + b_{\text{FC}} \tag{60}$$

Here $\bar{x}_{\text{true}}^{(L-1)}$ represents the true pooled features.

The recovered FC parameters absorb the accumulated scale factors from convolutional layers. The extraction recovers $W_{\text{rec}} = W_{\text{FC}} \cdot \text{diag}(\alpha^{(L-1)})$ rather than W_{FC} itself. Under this scale-only ambiguity, the recovered FC parameterization preserves the predicted labels for the evaluated inputs.

D. Pooling Layer Analysis

Pooling layers perform spatial downsampling of feature maps, reducing computational cost and providing translation invariance. The analysis below explains which pooling operations are compatible with cryptanalytic extraction.

1) Average Pooling: The output of average pooling is defined by Eq. 61.

$$h_{\text{out}}[i,j,c] = \frac{1}{k^2} \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} h_{\text{in}}[ki+p, kj+q, c]. \tag{61}$$

Here $h_{\text{out}}[i,j,c]$ denotes the output activation at spatial position (i,j) and channel c, computed as the mean activation within a $k \times k$ pooling window over the input h_{in}.

Linearity: The linear operation of average pooling is represented in Eq. 62.

$$\mathbf{h}_{\text{out}} = \mathbf{P}_{\text{avg}} \mathbf{h}_{\text{in}} \tag{62}$$

Here $\mathbf{P}_{\text{avg}}$ is a downsampling matrix that is block-diagonal with $(1/k^2)$ entries in each block.

Gradient flow: The gradient flow for average pooling during backpropagation is represented in Eq. 63.

$$\frac{\partial \mathcal{L}}{\partial h_{\text{in}}[ki+p, kj+q, c]} = \frac{1}{k^2} \frac{\partial \mathcal{L}}{\partial h_{\text{out}}[i, j, c]}. \tag{63}$$

Gradients flow uniformly to all input positions (p, q) within each pooling window centered at output position (i, j).

2) Max Pooling: The output of max pooling is defined by Eq. 64.

$$h_{\text{out}}[i, j, c] = \max_{p \in [0,k), q \in [0,k)} h_{\text{in}}[ki+p, kj+q, c]. \tag{64}$$

Here $h_{\text{out}}[i, j, c]$ denotes the output activation, which selects the maximum activation within each spatial window of the input h_{in}.

Non-linearity: Max pooling is a non-linear operation that cannot be expressed as a matrix multiplication. The output depends on the *location* of the maximum element, which varies across inputs.

Non-differentiability: The max operation is non-differentiable at points where multiple elements achieve the maximum value. This creates discontinuities in the gradient landscape.

Gradient flow: The gradient flow for max pooling is represented in Eq. 65.

$$\frac{\partial \mathcal{L}}{\partial h_{\text{in}}[ki+p, kj+q, c]} = \begin{cases} \frac{\partial \mathcal{L}}{\partial h_{\text{out}}[i,j,c]} & (p, q) = \text{argmax}_{p', q'} \, h_{\text{in}} \\ 0 & \text{otherwise} \end{cases} \tag{65}$$

Gradients flow exclusively through the maximum element (p, q) within each pooling window, and this selective gradient flow destroys information about non-maximal elements.

Incompatibility with extraction: Max pooling introduces several obstacles for the current extraction method:

- **Ill-conditioned signatures:** The signature matrix $\mathbf{S}$ becomes rank-deficient because non-maximal spatial locations are not observed through the pooling output. This violates the full-rank assumption required for SVD-based extraction.
- **Unstable boundary geometry:** The active max locations change with the input, so the local linear map changes discontinuously across nearby queries. This breaks the stable dual-point localization and consistent hyperplane constraints used by the method.
- **Input-dependent structure:** The location of maximum elements changes across different inputs, so the current method cannot construct a consistent linear system for weight extraction. The effective local Jacobian is input-dependent.
- **Information bottleneck:** Each pooling window discards $(k^2 - 1)/k^2$ of the spatial information. For 2×2 pooling, this represents 75% information loss, rendering signature recovery severely ill-conditioned.

Mathematical formulation: The Jacobian of a max pooling layer is defined by Eq. 66.

$$\mathbf{J}_{\text{max}} = \frac{\partial h_{\text{out}}}{\partial h_{\text{in}}}. \tag{66}$$

Here $\mathbf{J}_{\text{max}}$ is a sparse input-dependent matrix. For average pooling, $\mathbf{J}_{\text{avg}}$ is a fixed full-rank matrix independent of the input. For max pooling, $\mathbf{J}_{\text{max}}$ has rank at most hw/k^2, making the linear-system formulation for weight extraction ill-conditioned.

APPENDIX C
ALGORITHM PSEUDOCODE

The following pseudocode specifies the core algorithms underlying the cryptanalytic extraction method.

A. Complete CNN extraction algorithm

Algorithm 1 presents the end-to-end procedure for extracting all weights from a target CNN with hard-label oracle access.

Algorithm 1 Complete extraction

Require: Oracle $\mathcal{O}_f$, architecture, threshold τ
Ensure: Extracted network $\tilde{f}$

▷ Phase 1: Extract convolutional layers sequentially
for $i = 0, 1, \ldots, L_{\text{conv}} - 1$ **do**
  ▷ Collect dual points for layer i
  $\mathcal{D}^{(i)} \leftarrow \text{DUALPOINTFINDING}(\mathcal{O}_f, i, \{\tilde{f}_j\}_{j<i})$ ▷ Alg. 2
  ▷ Extract signatures for each channel
  for $c = 1, \ldots, C_{\text{out}}^{(i)}$ **do**
    if $i = 0$ **then**
      $(\tilde{w}_c, \tilde{b}_c) \leftarrow \text{SIGNATURELAYER0}(\mathcal{D}_c^{(i)})$ ▷ Alg. 3
    else
      $(\tilde{w}_c, \tilde{b}_c) \leftarrow \text{SIGNATUREHIDDEN}(\mathcal{D}_c^{(i)}, \{\tilde{f}_j\}_{j<i})$ ▷ Alg. 4
    end if
  end for
  ▷ Recover signs for layer i
  if $i = 0$ **then**
    $\{s_c\} \leftarrow \text{SIGNRECOVERYLAYER0}(\{\tilde{w}_c\}, \mathcal{O}_f, \tau)$ ▷ Alg. 6
  else
    $\{s_c\} \leftarrow \text{SIGNRECOVERYHIDDEN}(\cdots, i, L_{\text{conv}}, \tau)$ ▷ Alg. 7
  end if
  $\tilde{A}^{(i)} \leftarrow \{s_c \cdot \tilde{w}_c\}$, $\tilde{b}^{(i)} \leftarrow \{s_c \cdot \tilde{b}_c\}$
end for
▷ Phase 2: Extract fully-connected layer
$(\tilde{W}_{\text{FC}}, \tilde{b}_{\text{FC}}) \leftarrow \text{SIGNATUREFC}(\mathcal{O}_f, \{\tilde{f}_j\}_{j<L_{\text{conv}}})$ ▷ Alg. 5
$s_{\text{FC}} \leftarrow \text{SIGNRECOVERYFC}(\tilde{W}_{\text{FC}}, \tilde{b}_{\text{FC}}, \{\tilde{f}_j\}_{j<L_{\text{conv}}}, \mathcal{O}_f)$ ▷ Alg. 8
$\tilde{W}^{(L-1)} \leftarrow s_{\text{FC}} \cdot \tilde{W}_{\text{FC}}$, $\tilde{b}^{(L-1)} \leftarrow s_{\text{FC}} \cdot \tilde{b}_{\text{FC}}$
return $\tilde{f}$

B. Dual Point Finding

Algorithm 2 discovers dual points via boundary walking and binary search.

C. Signature Recovery Algorithms

1) Signature recovery for layer 0: Algorithm 3 extracts weight signatures for the first convolutional layer operating directly on input.

2) Signature recovery for hidden layers: Algorithm 4 extracts signatures for intermediate convolutional layers.

3) Signature recovery for FC layer: Algorithm 5 extracts the fully-connected output layer via least squares.

Algorithm 2 Dual point finding

Require: Hard-label oracle $\mathcal{O}_f$, target count m, subspace dimension κ, precision ε
Ensure: Dual point triples $\{(x_{\text{left}}^{(j)}, x_{\text{dual}}^{(j)}, x_{\text{right}}^{(j)})\}_{j=1}^{m}$

Sample x_0, x_1 with $\mathcal{O}_f(x_0) \neq \mathcal{O}_f(x_1)$
$x_b \leftarrow \textsc{BinarySearch}(x_0, x_1, \varepsilon)$ ▷ Locate boundary point
$\{v_1, \ldots, v_\kappa\} \leftarrow \textsc{RandomOrthonormalBasis}(\kappa)$
$\mathcal{D} \leftarrow \emptyset$
while $|\mathcal{D}| < m$ **do**
▷ Estimate boundary normal in subspace
  for $j = 1$ to κ **do**
    $\delta_j^+ \leftarrow \textsc{ForwardDistance}(x_b, v_j, \mathcal{O}_f)$
    $\delta_j^- \leftarrow \textsc{BackwardDistance}(x_b, v_j, \mathcal{O}_f)$
  end for
  $n_b \leftarrow \text{normalize}\left(\sum_{j=1}^{\kappa} \frac{\delta_j^- - \delta_j^+}{\delta_j^- + \delta_j^+} v_j\right)$
▷ Compute tangent direction and walk
  $r \sim \mathcal{N}(0, I_\kappa)$
  $t_b \leftarrow r - (r \cdot n_b) n_b$
  $\hat{t}_b \leftarrow t_b / \|t_b\|$
  $\tau^* \leftarrow \textsc{MaxStepSize}(x_b, \hat{t}_b, \mathcal{O}_f)$
  if $\tau^* > \tau_{\text{threshold}}$ **then**
    $x_{\text{mid}} \leftarrow x_b + \frac{\tau^*}{2} \hat{t}_b$
    $x_{\text{end}} \leftarrow x_b + \tau^* \hat{t}_b$
    $x_{\text{dual}} \leftarrow \textsc{BinarySearch}(x_{\text{end}}, x_{\text{end}} + \delta \hat{t}_b, \varepsilon)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_{\text{mid}}, x_{\text{dual}}, x_{\text{end}})\}$
    $x_b \leftarrow x_{\text{dual}}$
  else
    $x_b \leftarrow x_b + \tau_{\text{step}} \hat{t}_b$ ▷ Small exploration step
  end if
end while
return $\mathcal{D}$

Algorithm 3 Signature Recovery, Layer 0

Require: Dual points $\mathcal{D}_c = \{(x_{\text{left}}^{(j)}, x_{\text{dual}}^{(j)}, x_{\text{right}}^{(j)})\}$ for channel c
Ensure: Weight signature $\tilde{w}_c \in \mathbb{R}^{C_{\text{in}} \cdot k}$, bias $\tilde{b}_c \in \mathbb{R}$

▷ Build centered constraint matrix from dual points
$\mathcal{P}_c \leftarrow []$ ▷ Patch rows
for each $(x_{\text{left}}, x_{\text{dual}}, x_{\text{right}}) \in \mathcal{D}_c$ **do**
  $p \leftarrow$ spatial position of dual point
  patch $\leftarrow G_0(x_{\text{dual}})[:, p]$ ▷ Extract receptive field
  Append patch to $\mathcal{P}_c$
end for
$\bar{g}_c \leftarrow |\mathcal{P}_c|^{-1} \sum_{\text{patch} \in \mathcal{P}_c}$ patch
$D_c \leftarrow []$ ▷ Empty centered matrix
for each patch $\in \mathcal{P}_c$ **do**
  Append $(\text{patch} - \bar{g}_c)^\top$ as row to D_c
end for
▷ SVD to find null space
$U, \Sigma, V \leftarrow \text{SVD}(D_c)$
$\hat{w}_c \leftarrow V[:, -1]$ ▷ Right singular vector for $\varsigma_{\min}$
$\tilde{w}_c \leftarrow \hat{w}_c / \|\hat{w}_c\|_2$ ▷ Normalize weight
▷ Refine bias via median estimation
$\tilde{b}_c \leftarrow -\text{median}\{\tilde{w}_c^\top G_0(x_{\text{dual}}^{(j)})[:, p_j]\}_{j=1}^{|\mathcal{D}_c|}$
return $(\tilde{w}_c, \tilde{b}_c)$

Algorithm 4 Signature Recovery, Hidden Layers

Require: Dual points $\mathcal{D}_c$ for channel c, extracted layers $\{\tilde{f}_j\}_{j<i}$
Ensure: Weight signature $\tilde{w}_c \in \mathbb{R}^{C_{\text{in}}^{(i)} \cdot k^{(i)}}$, bias $\tilde{b}_c$

▷ Forward propagate through extracted layers
$\mathcal{P}_c \leftarrow []$
for each $(x_{\text{left}}, x_{\text{dual}}, x_{\text{right}}) \in \mathcal{D}_c$ **do**
  $h^{(i-1)} \leftarrow \tilde{f}_{i-1} \circ \cdots \circ \tilde{f}_0(x_{\text{dual}})$ ▷ Layer i input
  $p \leftarrow$ spatial position of dual point
  patch $\leftarrow G_i(h^{(i-1)})[:, p]$
  Append $(\text{patch}, h^{(i-1)}, p)$ to $\mathcal{P}_c$
end for
$\bar{g}_c \leftarrow |\mathcal{P}_c|^{-1} \sum_{(\text{patch}, h, p) \in \mathcal{P}_c}$ patch
$D_c \leftarrow []$
for each $(\text{patch}, h^{(i-1)}, p) \in \mathcal{P}_c$ **do**
  Append $(\text{patch} - \bar{g}_c)^\top$ as row to D_c
end for
▷ SVD extraction (same as Layer 0)
$U, \Sigma, V \leftarrow \text{SVD}(D_c)$
$\hat{w}_c \leftarrow V[:, -1]$
$\tilde{w}_c \leftarrow \hat{w}_c / \|\hat{w}_c\|_2$
$\tilde{b}_c \leftarrow -\text{median}\{\tilde{w}_c^\top G_i(h^{(i-1,j)})[:, p_j]\}$
return $(\tilde{w}_c, \tilde{b}_c)$

Algorithm 5 Signature Recovery, FC Layer

Require: Oracle $\mathcal{O}_f$, extracted conv layers $\{\tilde{f}_j\}$, reference class ℓ_{ref}
Ensure: FC weights $\tilde{W}_{\text{FC}} \in \mathbb{R}^{K \times d}$, biases $\tilde{b}_{\text{FC}} \in \mathbb{R}^K$

▷ Collect dual points between class pairs
$\mathcal{C} \leftarrow []$ ▷ Constraint equations
for each class pair (j, ℓ) with $j \neq \ell$ **do**
  Find dual points on boundary $\mathcal{B}_{j,\ell}$
  for each dual point x^* on $\mathcal{B}_{j,\ell}$ **do**
    $h \leftarrow \text{AvgPool}(\tilde{f}_{L_{\text{conv}}-1} \circ \cdots \circ \tilde{f}_0(x^*))$ ▷ FC input
    Add constraint: $(W_j - W_\ell)^\top h + (b_j - b_\ell) = 0$
  end for
end for
▷ Solve relative to reference class
Set $W_{\ell_{\text{ref}}} = 0$, $b_{\ell_{\text{ref}}} = 0$
$(\tilde{W}_{\text{FC}}, \tilde{b}_{\text{FC}}) \leftarrow \text{LeastSquares}(\mathcal{C})$
return $(\tilde{W}_{\text{FC}}, \tilde{b}_{\text{FC}})$

Algorithm 6 Sign Recovery, Input Layer

Require: Signatures $\{\tilde{w}_c\}_{c=1}^{C}$, dual points $\mathcal{D}$, oracle $\mathcal{O}_f$, margin threshold τ
Ensure: Signs $\{s_c\}_{c=1}^{C}$

▷ Stage 1: Voting via neuron wiggle
for $c = 1, \ldots, C$ **do**
  $n_+, n_- \leftarrow 0, 0$
  for each spatial position p **do**
    $v_{c,p} \leftarrow \textsc{Unfold}(\tilde{w}_c, p)$ ▷ Input-space perturbation
    $\nu_{c,p} \leftarrow \sum_{x_0, \epsilon} [\Delta(x_0, v_{c,p}, \epsilon) - \Delta(x_0, -v_{c,p}, \epsilon)]$
    if $\nu_{c,p} > 0$ **then** $n_+ \leftarrow n_+ + 1$
    else if $\nu_{c,p} < 0$ **then** $n_- \leftarrow n_- + 1$
    end if
  end for
  $\hat{s}_c \leftarrow \text{sign}(n_+ - n_-)$
  $\text{conf}_c \leftarrow 1 - 2\exp(-2(n_+ + n_-)(|n_+ - n_-|/(n_+ + n_-))^2)$
end for
▷ Stage 2: Discrete optimization for low-confidence channels
$\mathcal{C}_{\text{low}} \leftarrow \{c : \text{conf}_c < \tau\}$
if $|\mathcal{C}_{\text{low}}| > 0$ **then**
  best_svgr $\leftarrow \infty$
  for each $\mathbf{s}_{\text{low}} \in \{-1, +1\}^{|\mathcal{C}_{\text{low}}|}$ **do**
    Apply signs: $s_c \leftarrow s_c^{\text{low}}$ for $c \in \mathcal{C}_{\text{low}}$, else $s_c \leftarrow \hat{s}_c$
    Extract Layer 1 signatures using signed Layer 0
    svgr $\leftarrow$ mean SVGR of Layer 1 channels
    if svgr $<$ best_svgr **then**
      best_svgr $\leftarrow$ svgr
      $\mathbf{s}^* \leftarrow \mathbf{s}_{\text{low}}$
    end if
  end for
  Update $\{s_c\}$ with $\mathbf{s}^*$ for $c \in \mathcal{C}_{\text{low}}$
end if
return $\{s_c\}_{c=1}^{C}$

D. Sign Recovery Algorithms

1) Sign recovery for input layer: Algorithm 6 recovers signs for the input layer using the two-step neuron wiggle procedure with SVGR-guided discrete optimization.

2) Sign recovery for hidden layers: Algorithm 7 gives hidden-layer sign recovery.

3) Sign recovery for output layer: Algorithm 8 determines the global sign of the fully-connected output layer.

APPENDIX D
EXPERIMENTAL SUPPLEMENTARY DETAILS

A. Evaluation Metrics

The evaluation uses metrics for directional accuracy, constraint matrix conditioning, and prediction agreement.

Algorithm 7 Sign Recovery, Hidden Layers

Require: Signatures $\{\tilde{w}_c\}$, extracted layers $\{\tilde{f}_j\}_{j<i}$, oracle $\mathcal{O}_f$, layer index i, convolutional layer count L_{conv}, margin threshold τ
Ensure: Signs $\{s_c\}_{c=1}^{C}$

▷ Stage 1: Voting via neuron wiggle
for $c = 1, \ldots, C$ **do**
  $n_+, n_- \leftarrow 0, 0$
  for each spatial position p **do**
    ▷ Backpropagate perturbation to input space
    $\mathcal{J}^{(i)} \leftarrow$ Jacobian of $\tilde{f}_{i-1} \circ \cdots \circ \tilde{f}_0$ at x_0
    $v_{c,p} \leftarrow [\mathcal{J}^{(i)}]^\top \cdot \text{UNFOLD}(\tilde{w}_c, p)$
    $\nu_{c,p} \leftarrow \sum_{x_0,\epsilon}[\Delta(x_0, v_{c,p}, \epsilon) - \Delta(x_0, -v_{c,p}, \epsilon)]$
    if $\nu_{c,p} > 0$ **then** $n_+ \leftarrow n_+ + 1$
    else if $\nu_{c,p} < 0$ **then** $n_- \leftarrow n_- + 1$
    end if
  end for
  $\hat{s}_c \leftarrow \text{sign}(n_+ - n_-)$
  $\text{margin}_c \leftarrow |n_+ - n_-|/(n_+ + n_-)$
end for
▷ Stage 2: Layer-specific discrete optimization
$\mathcal{C}_{\text{low}} \leftarrow \{c : \text{margin}_c < \tau\}$
if $|\mathcal{C}_{\text{low}}| > 0$ **then**
  best_score $\leftarrow \infty$
  for each $\mathbf{s}_{\text{low}} \in \{-1, +1\}^{|\mathcal{C}_{\text{low}}|}$ **do**
    Apply candidate signs to layer i
    if $i < L_{\text{conv}} - 1$ **then**
      Extract Layer $i + 1$ signatures
      score $\leftarrow$ mean SVGR_{i+1}
    else
      Extract FC layer and set score $\leftarrow$ FC least-squares residual
    end if
    if score $<$ best_score **then**
      best_score $\leftarrow$ score
      $\mathbf{s}^* \leftarrow \mathbf{s}_{\text{low}}$
    end if
  end for
  Update $\{s_c\}$ with $\mathbf{s}^*$ for $c \in \mathcal{C}_{\text{low}}$
end if
return $\{s_c\}_{c=1}^{C}$

Algorithm 8 Sign Recovery, Output Layer

Require: FC weights $\tilde{W}_{\text{FC}}$, biases $\tilde{b}_{\text{FC}}$, extracted convolutional layers $\{\tilde{f}_j\}_{j<L_{\text{conv}}}$, oracle $\mathcal{O}_f$, test set $\mathcal{X}_{\text{test}}$
Ensure: Global sign $s_{\text{FC}} \in \{-1, +1\}$

$\text{PF}_{+1} \leftarrow 0$
$\text{PF}_{-1} \leftarrow 0$
for each $x \in \mathcal{X}_{\text{test}}$ **do**
  $h \leftarrow \text{AvgPool}(\tilde{f}_{L_{\text{conv}}-1} \circ \cdots \circ \tilde{f}_0(x))$
  $\ell_{+1} \leftarrow (+1) \cdot \tilde{W}_{\text{FC}} h + (+1) \cdot \tilde{b}_{\text{FC}}$
  $\ell_{-1} \leftarrow (-1) \cdot \tilde{W}_{\text{FC}} h + (-1) \cdot \tilde{b}_{\text{FC}}$
  $\hat{y}_{+1} \leftarrow \arg\max_k \ell_{+1}[k]$
  $\hat{y}_{-1} \leftarrow \arg\max_k \ell_{-1}[k]$
  $y \leftarrow \mathcal{O}_f(x)$
  if $\hat{y}_{+1} = y$ **then** $\text{PF}_{+1} \leftarrow \text{PF}_{+1} + 1$
  end if
  if $\hat{y}_{-1} = y$ **then** $\text{PF}_{-1} \leftarrow \text{PF}_{-1} + 1$
  end if
end for
if $\text{PF}_{+1} > \text{PF}_{-1}$ **then**
  return $+1$
else
  return -1
end if

1) Cosine Similarity: Cosine similarity measures directional accuracy of extracted weights. The cosine similarity is defined by Eq. 67.

$$\text{Cos-Sim}(w_{\text{ext}}, w_{\text{true}}) = \frac{w_{\text{ext}}^\top w_{\text{true}}}{\|w_{\text{ext}}\|_2 \|w_{\text{true}}\|_2} \tag{67}$$

Here w_{ext} is the extracted weight and w_{true} is the ground truth weight.

Values near $+1$ or -1 indicate accurate direction recovery. The sign ambiguity from SVD extraction causes the ± 1 equivalence. Absolute cosine similarity |Cos-Sim| is reported when sign is unresolved.

2) SVGR: SVGR quantifies constraint matrix conditioning. It is defined by Eq. 68.

$$\text{SVGR} = \frac{\varsigma_{\min}}{\varsigma_{\min+1}} \tag{68}$$

Here $\varsigma_{\min}$ is the smallest singular value and $\varsigma_{\min+1}$ is the second smallest.

Small SVGR indicates clear separation between null space and column space. Values below 10^{-6} indicate reliable signature extraction. Large SVGR suggests insufficient or inconsistent constraints.

3) Prediction Fidelity: Prediction fidelity measures agreement between the extracted and original models on an evaluation set. Prediction fidelity is defined by Eq. 69.

$$\text{PF} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbb{1}\left[\mathcal{O}_{\tilde{f}}(x_i) = \mathcal{O}_f(x_i)\right] \tag{69}$$

Here N_{test} is the test set size and $\mathbb{1}[\cdot]$ is the indicator function.

Prediction fidelity of 1.0 indicates exact agreement on the sampled inputs. This metric directly measures extraction success on held-out data. Prediction fidelity is reported on both in-distribution and out-of-distribution evaluation sets.

B. Detailed Experimental Results

Per-channel extraction results are reported for all layers of the target CNN architecture. These data characterize extraction behavior across convolutional and fully-connected layers in the evaluated model.

1) Layer 0: First Convolutional Layer: Layer 0 consists of 8 convolutional filters (1×3 kernels) operating on the input image. The layer processes 3,136 total clusters (spatial positions times channels) with 392 unique positions per channel.

a) Signature Recovery.: Table XIV presents detailed signature extraction results including dual point statistics and SVD analysis for each channel.

TABLE XIV
LAYER 0 SIGNATURE DIAGNOSTICS. COLUMNS REPORT DUAL POINTS, POSITIONS, SVGR, AND BIAS.

Ch	Dual Pts	Samples	Positions	$\varsigma_{\min}$	$\varsigma_{\min+1}$	SVGR	Bias
0	18,125	200	4	7.52×10^{-8}	8.31	9.05×10^{-9}	-0.1634
1	18,208	200	5	1.47×10^{-6}	9.57	1.53×10^{-7}	$+0.1286$
2	11,242	200	7	9.21×10^{-6}	9.00	1.02×10^{-6}	-0.8790
3	13,234	200	7	1.60×10^{-7}	9.20	1.74×10^{-8}	-0.8198
4	18,136	200	4	6.01×10^{-6}	7.90	7.61×10^{-7}	$+0.2085$
5	17,963	200	5	5.91×10^{-7}	9.81	6.02×10^{-8}	$+0.2286$
6	8,933	200	9	3.30×10^{-6}	8.92	3.70×10^{-7}	$+1.0935$
7	7,054	200	12	3.12×10^{-6}	6.78	4.60×10^{-7}	$+1.4492$

Notes. 'Dual Pts' is total dual points collected. 'Samples' is points used for SVD. 'Positions' is distinct spatial positions sampled. SVGR is $\varsigma_{\min}/\varsigma_{\min+1}$.

b) Sign Recovery.: The sign recovery for Layer 0 employs the two-step neuron wiggle procedure with SVGR-guided discrete optimization. Channels 1, 3, and 4 exhibited low voting confidence (< 0.9999) and were resolved by

optimizing over $2^3 = 8$ retained sign combinations. Tables XV and XVI provide the detailed voting and discrete-optimization statistics. The final recovered signs for Layer 0 are $[+1, +1, +1, -1, -1, +1, -1, +1]$, achieving 100% accuracy against ground truth.

TABLE XV
LAYER 0 SIGN VOTING DIAGNOSTICS. VOTES AND CONFIDENCE SET OPTIMIZATION CANDIDATES.

Ch	Pos Votes	Neg Votes	Ratio	Conf	Init Vote	Correct?
0	846	529	61.5%	1.000	$+1$	Yes
1	781	919	54.1%	0.993	-1	**No**
2	1484	392	79.1%	1.000	$+1$	Yes
3	885	1005	53.2%	0.956	-1	Yes
4	863	802	51.8%	0.346	$+1$	**No**
5	1034	757	57.7%	1.000	$+1$	Yes
6	317	1426	81.8%	1.000	-1	Yes
7	1655	298	84.7%	1.000	$+1$	Yes

TABLE XVI
LAYER 0 DISCRETE OPTIMIZATION. LAYER 1 SVGR SELECTS THE RECOVERED SIGNS.

Ch 1	Ch 3	Ch 4	Layer 1 SVGR
-1	-1	-1	5.998×10^{-1}
-1	-1	$+1$	6.943×10^{-1}
-1	$+1$	-1	5.033×10^{-1}
-1	$+1$	$+1$	5.641×10^{-1}
$+1$	-1	-1	$\mathbf{5.728 \times 10^{-7}}$
$+1$	-1	$+1$	5.877×10^{-1}
$+1$	$+1$	-1	4.163×10^{-1}
$+1$	$+1$	$+1$	5.272×10^{-1}

2) Layer 1: Second Convolutional Layer: Layer 1 applies 8 convolutional filters (8×3 kernels) to the output of Layer 0, with 1,568 total clusters.

a) Signature Recovery.: Table XVII presents detailed signature extraction results for Layer 1.

TABLE XVII
LAYER 1 SIGNATURE DIAGNOSTICS. COLUMNS REPORT DUAL POINTS, POSITIONS, SVGR, AND BIAS.

Ch	Dual Pts	Samples	Positions	$\varsigma_{\min}$	$\varsigma_{\min+1}$	SVGR	Bias
0	9,753	9,049	196	2.81×10^{-5}	2.84	9.90×10^{-6}	$+0.0577$
1	10,937	9,409	196	1.19×10^{-4}	3.11	3.83×10^{-5}	-0.0275
2	11,110	9,509	196	3.09×10^{-4}	3.36	9.20×10^{-5}	-0.1068
3	12,134	9,671	196	2.30×10^{-5}	2.78	8.27×10^{-6}	-0.0083
4	10,736	9,398	196	3.51×10^{-5}	2.61	1.35×10^{-5}	$+0.2341$
5	2,303	2,303	189	2.88×10^{-5}	0.98	2.95×10^{-5}	$+0.0451$
6	10,967	9,406	196	3.34×10^{-5}	3.01	1.11×10^{-5}	-0.2347
7	10,532	9,267	196	9.52×10^{-5}	2.59	3.68×10^{-5}	$+0.2066$

b) Sign Recovery.: Table XVIII presents the voting-based sign recovery results for Layer 1. Channel 3 exhibited low confidence below the 0.9999 threshold and was resolved by optimizing over $2^1 = 2$ retained sign combinations using Layer 2's SVGR as the selection criterion.

Table XIX presents the discrete-optimization results for Layer 1's low-confidence channel. The SVGR of Layer 2 signature extraction serves as the selection criterion, and lower SVGR indicates better extraction quality and thus more likely correct signs.

TABLE XVIII
LAYER 1 SIGN VOTING DIAGNOSTICS. VOTES AND CONFIDENCE SET OPTIMIZATION CANDIDATES.

Ch	+Pos	−Pos	Ratio	Conf	Init Sign	Correct?
0	34	134	79.8%	1.000	-1	Yes
1	48	119	71.3%	1.000	-1	Yes
2	133	46	74.3%	1.000	$+1$	Yes
3	76	90	54.2%	-0.108	-1	Yes
4	21	157	88.2%	1.000	-1	Yes
5	175	16	91.6%	1.000	$+1$	Yes
6	147	33	81.7%	1.000	$+1$	Yes
7	33	144	81.4%	1.000	-1	Yes

Notes. Only channel 3 exhibits low confidence (-0.108), and all initial votes are correct.

TABLE XIX
LAYER 1 DISCRETE OPTIMIZATION. LAYER 2 SVGR SELECTS THE RECOVERED SIGN.

Ch 3 Sign	Layer 2 SVGR
-1	$\mathbf{6.078 \times 10^{-7}}$
$+1$	7.208×10^{-1}

Notes. The SVGR difference (6.08×10^{-7} vs. 0.72) selects -1 as the recovered sign.

Final recovered signs are $[-1, -1, +1, -1, -1, +1, +1, -1]$, achieving 100% accuracy.

3) Layer 2: Third Convolutional Layer: Layer 2 consists of 4 convolutional filters (8×3 kernels) with 392 total clusters.

a) Signature Recovery.: Table XX presents detailed signature extraction results for Layer 2.

TABLE XX
LAYER 2 SIGNATURE DIAGNOSTICS. COLUMNS REPORT DUAL POINTS, POSITIONS, SVGR, AND BIAS.

Ch	Dual Pts	Samples	Positions	$\varsigma_{\min}$	$\varsigma_{\min+1}$	SVGR	Bias
0	2,927	2,927	98	1.04×10^{-5}	3.39	3.08×10^{-6}	$+0.1036$
1	5,198	4,705	98	1.08×10^{-5}	4.83	2.24×10^{-6}	-0.0129
2	4,533	4,334	98	1.13×10^{-5}	5.04	2.24×10^{-6}	-0.0018
3	3,324	3,323	98	1.63×10^{-5}	2.34	6.96×10^{-6}	$+0.0329$

b) Sign Recovery.: Table XXI presents the voting-based sign recovery results for Layer 2. Two channels (0, 2) exhibited low confidence below the 0.9999 threshold and were resolved by optimizing over $2^2 = 4$ retained sign combinations using FC solvability as the selection criterion.

TABLE XXI
LAYER 2 SIGN VOTING DIAGNOSTICS. VOTES AND CONFIDENCE SET OPTIMIZATION CANDIDATES.

Ch	+Pos	−Pos	Ratio	Conf	Init Sign	Correct?
0	60	27	69.0%	0.996	$+1$	Yes
1	82	12	87.2%	1.000	$+1$	Yes
2	33	55	62.5%	0.872	-1	Yes
3	10	79	88.8%	1.000	-1	Yes

Notes. All initial votes are correct, and discrete optimization serves as validation.

Table XXII presents the discrete-optimization results for Layer 2's low-confidence channels. Unlike earlier layers, Layer 2 uses FC solvability as the selection criterion since it is the final convolutional layer.

TABLE XXII
LAYER 2 DISCRETE OPTIMIZATION. FC SOLVABILITY SELECTS THE RECOVERED SIGNS.

Ch 0	Ch 2	Selected by FC solv.
-1	-1	No
-1	$+1$	No
$+1$	-1	**Yes**
$+1$	$+1$	No

Notes. The selected combination $(+1, -1)$ gives the best FC least-squares solvability among the retained candidates, matching the initial voting results.

Final recovered signs are $[+1, +1, -1, -1]$, achieving 100% accuracy.

4) Fully-Connected Output Layer: The FC layer maps 64 flattened features (from $4 \times 16 = 64$ spatial positions of Layer 2 output) to 10 output classes. Extraction uses the least squares formulation with class 2 as reference.

a) Least Squares Extraction.: Table XXIII presents per-class extraction results for the FC layer.

TABLE XXIII
FC EXTRACTION BY CLASS. COSINE SIMILARITY REPORTS PARAMETER RECOVERY QUALITY.

Class	$\|W\|$	Bias	Raw CS	Adj CS
0	14,997.72	$+24.71$	0.9452	1.0000
1	20,937.07	-37.05	0.9448	1.0000
2	0.00	0.00	(reference)	N/A
3	12,919.66	-31.82	0.9507	1.0000
4	23,255.45	$+14.95$	0.9449	1.0000
5	14,271.93	-22.04	0.9402	1.0000
6	15,761.59	-6.15	0.9403	1.0000
7	17,867.41	-10.20	0.9374	1.0000
8	16,393.20	$+2.40$	0.9401	1.0000
9	18,864.79	$+1.76$	0.9383	1.0000

b) FC Sign Recovery via Prediction Fidelity.: The FC layer sign is determined by testing both sign configurations on held-out test samples and measuring prediction fidelity. Table XXIV presents the prediction-fidelity comparison for FC layer sign recovery.

TABLE XXIV
FC GLOBAL SIGN RECOVERY. PREDICTION FIDELITY SELECTS THE SIGN.

Global Sign	PF	Matches	Selected
$+1$	0.0000	0/1,000	No
-1	1.0000	1,000/1,000	**Yes**

The prediction-fidelity difference (100% vs. 0%) selects the global sign -1.

TABLE XXV
OVERALL EXTRACTION SUMMARY. METRICS AGGREGATE SVGR, COSINE SIMILARITY, AND SIGN ACCURACY.

Layer	Ch	Mean SVGR	Raw CS	Adj CS	Sign Acc.
Layer 0	8	3.79×10^{-7}	1.0000	1.0000	8/8 (100%)
Layer 1	8	2.85×10^{-5}	0.9109	1.0000	8/8 (100%)
Layer 2	4	3.63×10^{-6}	0.9143	1.0000	4/4 (100%)
FC	9	N/A	0.9424	1.0000	1/1 (100%)
Total	**29**	N/A	N/A	**1.0000**	**21/21 (100%)**

C. Statistical Summary

Table XXV summarizes extraction quality across all layers.

Remark (Large Bias Error in Deeper Layers): The reported bias errors for Layer 1 (~ 0.56) and Layer 2 (~ 0.82) appear large compared with Layer 0 ($\sim 10^{-8}$). This is an artifact of the bias-error metric rather than a failure of extraction accuracy. The bias_err metric compares the recovered bias $\tilde{b}$ with the true bias b without accounting for scale factors. SVD-based signature recovery produces unit-norm weight vectors, while the true weights have arbitrary norms. The recovered bias is therefore $\tilde{b} = b/\alpha$, where α is the unknown scale factor. For deeper layers, the effective scale α_{eff} grows larger, causing direct bias comparison to overstate the error. The FC layer absorbs this scale discrepancy during extraction, as described in Section B-C3. The final extracted network still achieves 100% prediction fidelity on the reported evaluation sets.

D. Sign Vote Sensitivity Analysis

The sign-vote ablation varies samples per position and reports correct signs, overall accuracy, and mean confidence. Table XXVI summarizes the sensitivity results for Layer 1. With one perturbation sample per position, sign recovery achieves 75% accuracy with mean confidence 0.542. Increasing to 3 samples per position achieves 100% accuracy with confidence 0.781. Five samples per position maintains 100% accuracy while improving confidence to 0.862. Further increasing to 10 or 20 samples provides marginal confidence gains at additional query cost. In this ablation, 3 to 5 samples per position balance accuracy and query cost.

TABLE XXVI
SIGN VOTE SAMPLE ABLATION. SAMPLES PER POSITION AFFECT ACCURACY AND CONFIDENCE.

Samples/Position	Correct Signs	Accuracy	Mean Confidence
1 sample	6 / 8	75.0%	0.542
3 samples	8 / 8	100.0%	0.781
5 samples	8 / 8	100.0%	0.862
10 samples	8 / 8	100.0%	0.894
20 samples	8 / 8	100.0%	0.921

E. Per-Class Prediction-Fidelity Evaluation Details

This appendix gives per-class accuracy and prediction-fidelity breakdowns for the end-to-end extraction evaluation.

1) Training Set Per-Class Results: Table XXVII presents the per-class breakdown of classification accuracy and prediction fidelity on the MNIST training set (60,000 samples).

TABLE XXVII
MNIST TRAIN FIDELITY BY CLASS. EXTRACTED PREDICTIONS MATCH THE ORIGINAL MODEL.

Class	Total	Orig Correct	Orig Acc	Ext Acc	PF Match	PF
0	5,923	5,681	95.91%	95.91%	5,923/5,923	100.00%
1	6,742	6,533	96.90%	96.90%	6,742/6,742	100.00%
2	5,958	4,786	80.33%	80.33%	5,958/5,958	100.00%
3	6,131	4,661	76.02%	76.02%	6,131/6,131	100.00%
4	5,842	5,539	94.81%	94.81%	5,842/5,842	100.00%
5	5,421	3,691	68.09%	68.09%	5,421/5,421	100.00%
6	5,918	5,342	90.27%	90.27%	5,918/5,918	100.00%
7	6,265	5,519	88.09%	88.09%	6,265/6,265	100.00%
8	5,851	5,164	88.26%	88.26%	5,851/5,851	100.00%
9	5,949	5,328	89.56%	89.56%	5,949/5,949	100.00%
Total	**60,000**	**52,244**	**87.07%**	**87.07%**	**60,000/60,000**	**100.00%**

The per-class accuracy varies across digit classes, ranging from 68.09% (class 5) to 96.90% (class 1). This variation reflects the inherent difficulty of classifying certain digit shapes with the target CNN architecture. Every class achieves 100.00% prediction fidelity on the sampled training inputs, indicating that the extracted model matches the original model on these inputs regardless of classification difficulty.

2) Test Set Per-Class Results: Table XXVIII presents the per-class breakdown on the MNIST test set (10,000 samples).

TABLE XXVIII
MNIST TEST FIDELITY BY CLASS. EXTRACTED PREDICTIONS MATCH THE ORIGINAL MODEL.

Class	Total	Orig Correct	Orig Acc	Ext Acc	PF Match	PF
0	980	941	96.02%	96.02%	980/980	100.00%
1	1,135	1,104	97.27%	97.27%	1,135/1,135	100.00%
2	1,032	840	81.40%	81.40%	1,032/1,032	100.00%
3	1,010	774	76.63%	76.63%	1,010/1,010	100.00%
4	982	937	95.42%	95.42%	982/982	100.00%
5	892	623	69.84%	69.84%	892/892	100.00%
6	958	863	90.08%	90.08%	958/958	100.00%
7	1,028	893	86.87%	86.87%	1,028/1,028	100.00%
8	974	877	90.04%	90.04%	974/974	100.00%
9	1,009	900	89.20%	89.20%	1,009/1,009	100.00%
Total	**10,000**	**8,752**	**87.52%**	**87.52%**	**10,000/10,000**	**100.00%**

The test set per-class accuracy closely mirrors the training set distribution, with classes 1 and 4 achieving highest accuracy (>95%) and classes 3 and 5 exhibiting lowest accuracy (<77%). This consistency between training and test accuracy is consistent with generalization beyond the training set. The observed 100.00% prediction fidelity across all test classes indicates that the extracted model matches the original model on the sampled unseen test inputs.

3) Random Data Per-Class Results: Table XXIX presents the per-class prediction-fidelity breakdown on random noise inputs (10,000 samples drawn uniformly from $[0, 1]^{784}$).

The random input classification is heavily skewed toward certain classes (primarily 8 and 9), reflecting the structure of the learned decision boundaries in high-dimensional space. Random uniform noise tends to activate certain decision regions more frequently due to the geometry of ReLU polytopes. Despite this non-uniform distribution, the extracted model achieves 100.00% prediction fidelity on all sampled random inputs, providing an out-of-distribution check of the recovered decision function.

TABLE XXIX
RANDOM INPUT FIDELITY BY CLASS. COUNTS FOLLOW ORIGINAL PREDICTIONS ON RANDOM PROBES.

Predicted Class	Total	PF Match	PF
0	187	187/187	100.00%
4	890	890/890	100.00%
6	1,286	1,286/1,286	100.00%
7	9	9/9	100.00%
8	2,763	2,763/2,763	100.00%
9	4,865	4,865/4,865	100.00%
Total	**10,000**	**10,000/10,000**	**100.00%**