Scalable Scan-Chain-Based Extraction of Neural Network Models

Shui Jiang
The Chinese University of Hong Kong
N.T., Hong Kong
sjiang22@cse.cuhk.edu.hk

Seetal Potluri
North Carolina State University
NC, USA
spotlur2@ncsu.edu

Tsung-Yi Ho
The Chinese University of Hong Kong
N.T., Hong Kong
tyho@cse.cuhk.edu.hk

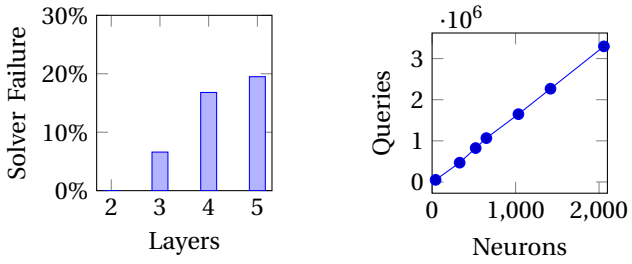
***Abstract*—Scan chains have greatly improved hardware testability while introducing security breaches for confidential data. Scan-chain attacks have extended their scope from cryptoprocessors to AI edge devices. The recently proposed scan-chain-based neural network (NN) model extraction attack (ICCAD 2021) made it possible to achieve fine-grained extraction and is multiple orders of magnitude more efficient both in queries and accuracy than its coarse-grained mathematical counterparts. However, both query formulation complexity and *constraint solver failures* increase drastically with network depth/size. We demonstrate a more powerful adversary, who is capable of improving scalability while maintaining accuracy, by relaxing high-fidelity constraints to formulate an approximate-fidelity-based layer-constrained least-squares extraction using *random queries*. We conduct our extraction attack on neural network inference topologies of different depths and sizes, targeting the `MNIST` digit recognition task. The results show that our method outperforms the scan-chain attack proposed in ICCAD 2021 by an average increase in the extracted neural network's functional accuracy of $\approx 32\%$ and $2-3$ orders of reduction in queries. Furthermore, we demonstrated that our attack is highly effective even in the presence of countermeasures against adversarial samples.**

***Index Terms*—Deep neural network, testing, scan-chain, model extraction, scalability**

I. Introduction

Machine Learning (ML) *model extraction* poses two significant challenges: (a) the adversary performs model extraction with the motivation to exploit the commercial value of the ML model IP [1]–[6]; and (b) the adversary uses the extracted model to craft superior adversarial samples [7]–[9]. These *coarse-grained model extraction* works assume an ML-as-a-Service application with the prediction application programmer interface and publicly accessible query interfaces facilitated by the cloud provider.

Recently, the neural network (NN) scan-chain attack in ICCAD 2021 [10] has exposed a new and powerful threat vector for edge devices. These attacks perform *fine-grained model extraction* by exploiting the scan-chain infrastructure that allows in-field testing [11]. For each query (input vector to the NN), the NN hardware accelerator is configured to execute the classification task using a *fast (system) clock*, and subsequently, the results of interest are captured at activation registers using a *slow (shift) clock* [10]. This scan-chain-based extraction was shown to defeat all the prior *coarse-grained* counterparts by multiple orders of magnitude both in terms of query count and extraction errors. However, solver failures



(a) *Constraint solver failures.* (b) *Queries.*

Fig. 1: Increase in *constraint solver failures* and *queries* with network depth and neurons respectively, for NNs targeting `MNIST` dataset, with *high-fidelity* scan-chain extraction.

as well as queries increase with network depth/size as shown for NNs targeting `MNIST` dataset in Figure 1.

This paper demonstrates a more powerful adversary, who is capable of improving scalability while maintaining accuracy. The adversary relaxes these constraints and exploits parallelism to formulate an *approximate-fidelity-based layer-constrained least-squares* attack using *random queries*. The main contributions of this paper are:

- We recognize the prior work in ICCAD 2021 [10] is limited to exploiting only one input per query (the input vector). We thereby exploit multiple inputs per query, effectively increasing parallelism, and hence scalability;
- We further improve scalability by relaxing high-fidelity constraints without loss in *functional accuracy*, and using layer-constrained *least-squares regression* to solve for the model parameters using *random queries*;
- Application on 14 different neural network architectures with varying neurons per layer and up to 5 layers, has resulted in $\approx 32\%$ improvement in accuracy on average and multiple orders of reduction in queries;
- We also show the efficacy of the proposed extraction attack in the presence of adversarial input defense schemes e.g. `FeatureSqueezing` [12] and `AdaptiveDenoising` [13]; and
- We finally demonstrate how to exploit the least-squares formulation to trade-off queries with the functional accuracy of the extracted model.

The paper is organized as follows: Sections II and III provide the threat model and background on NNs. Section IV explains the proposed scalable methodology on how to

978-3-9819263-7-8/DATE23/© 2023 EDAA

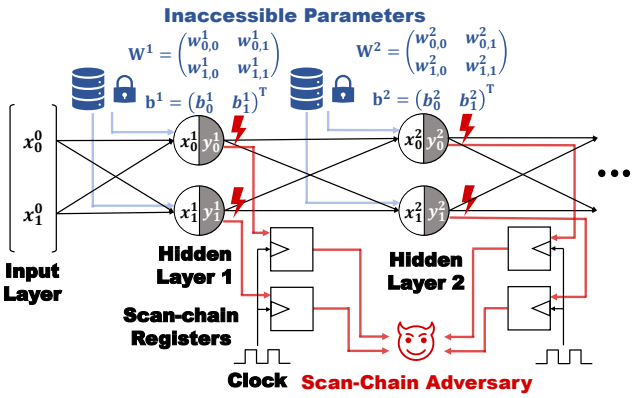


Fig. 2: Scan chains leaking neuron states after activation, in a fully connected neural network (FCNN).

exploit multiple inputs per query (the input vector), and the *least-squares regression* based extraction (LSR) in detail. Section V provides the results obtained by applying the proposed method on a wide range of NN architectures of varying depths and sizes, as well as comparisons with prior work. Subsequently, Section VI provides a discussion on the extraction performance in the presence of adversarial input defense schemes, and finally Section VII concludes the paper.

II. Threat Model

We follow the same threat model as laid out in the original neural network (NN) scan-chain attack from ICCAD 2021 [10]. We consider an adversary located at an outsourced assembly and test (OSAT) facility [14], having physical access to an edge device that contains the NN models. Their system-level-test (SLT) [14] feature translates to the adversary's capability to execute the classification for a certain number of functional cycles, switch to debug mode, and use the hardware debug port to leak internal states as shown in Figure 2. The adversary then performs a mathematical analysis on these internal states to extract the NN model parameters.

Recent work categorizes model extraction attacks into high-fidelity vs. high-accuracy [6]: prior scan-chain-based extraction [10] performs a *high-fidelity* attack that aims for general agreement between the extracted and victim models on any input vector. As discussed, this leads to an increase in query formulation complexity and solver conflicts, leading to *no solutions*. Our more powerful adversary addresses this issue by performing a *high-accuracy* attack by relaxing the *high-fidelity* constraints while maintaining accuracy.

Following the earlier work on *high-fidelity* scan-chain-based model extraction [10], we assume the adversary has (a) *full architecture knowledge*, including the number of layers, neurons in each layer, and the activation function; and (b) *complete vector output of every hidden layer*: under the scan-chain threat model, the classifier's response to each input query contains the vector output of an entire layer of neurons. For any given query, the intermediate outputs of different neurons in different layers will eventually be acquired from the hardware debug port by the adversary. Meanwhile, we assume that the NN parameters (weights and biases) are kept in private storage that is out of direct reach of scan chains.

III. Background

A fully connected neural network (FCNN) is a feed-forward neural network, consisting of fully connected layers that connect each neuron in one layer to every neuron in the succeeding layer. If we define $\mathbf{x}^i$, $\mathbf{y}^i$, $\mathbf{W}^i$, and $\mathbf{b}^i$ as the pre-activation output vector, output vector, weight matrix, and bias vector respectively of i^{th} hidden layer ($i > 0$), corresponding to NN input vector (query) $\mathbf{x}^0$, we have:

$$\mathbf{x}^i = \mathbf{W}^i \cdot \mathbf{y}^{(i-1)} + \mathbf{b}^i \tag{1}$$

$$\mathbf{y}^i = h(\mathbf{x}^i) \tag{2}$$

where $h(\cdot)$ is the non-linear activation function. Figure 2 shows an FCNN's two hidden layers. The weights, biases, and activation values are represented as double-precision floating-point numbers in this work. The notation a_0-a_1-a_2-…-a_n is used in this paper, which signifies that the network has a_0 elements in the input vector, and a_i neurons in the i^{th} hidden layer ($i > 0$). The `Sigmoid` is the non-linear activation function $h(\cdot)$ we used. The `Sigmoid` activation function is given below, where x is the pre-activation result of a neuron and $h(x)$ is the activation result:

$$h(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

The output layer uses *argmax* to select the neuron index with the largest output of the penultimate layer as the output of the classifier.

IV. Attack Overview

In this section, we present the detailed crafting of our attack. We first identify the limitations of the existing approach [10] and then present the details of our more powerful adversary, the systematic approach used to model the problem, and finally how to leverage least-squares regression to extract the model parameters with high accuracy.

A. Motivation

The state-of-the-art scan-chain-based extraction uses *isolated activation* [10] to extract weights and biases of a neural network, which consists of two steps: (a) *Q-point* search; and (b) application of small perturbations around the *Q-point* to expose the model parameters. Although this method reduces queries by many orders of magnitude compared to its mathematical extraction counterparts [4]–[6], each query is still greatly underutilized. The adversary has to apply at least two queries (let alone the additional queries used to locate a *Q-point*) and inspect the scan outputs for all the neurons in each layer only to deduce one parameter, which makes the method not scalable to deeper/larger networks with millions of parameters.

Additionally, query formulation becomes increasingly complex with growth in network depth/size, thereby increasingly causing failures when trying to solve the isolated activation constraints, as illustrated earlier in Figure 1a. As a result, it is important to come up with a simpler querying methodology to address the formulation complexity issue, or in other words, make the extraction more *scalable*. Therefore, for every input

query, we propose to make full use of the outputs from every neuron in every layer. We collect the intermediate output vectors from every layer to form linear systems and solve the parameters using *least-squares regression*.

B. Modeling

Due to the *bijective* property of the `Sigmoid` activation function, from Equation 2 we have $\mathbf{x}^i = h^{-1}(\mathbf{y}^i)$. Substituting this in Equation 1 (corresponding to a single query), we have:

$$h^{-1}(\mathbf{y}^i) = \begin{bmatrix} \mathbf{W}^i & \mathbf{b}^i \end{bmatrix} \cdot \begin{bmatrix} \mathbf{y}^{(i-1)} \\ 1 \end{bmatrix} \tag{4}$$

When we introduce N queries, the intermediate layers' output vectors can be stacked together. For the sake of simplicity, for any given layer i, if we define:

$$\mathbf{P}^i = \begin{bmatrix} h^{-1}(\mathbf{y}^i(0)) & h^{-1}(\mathbf{y}^i(1)) & \dots & h^{-1}(\mathbf{y}^i(N-1)) \end{bmatrix} \tag{5}$$

$$\mathbf{A}^i = \begin{bmatrix} \mathbf{W}^i & \mathbf{b}^i \end{bmatrix} \tag{6}$$

$$\mathbf{Q}^i = \begin{bmatrix} \mathbf{y}^{(i-1)}(0) & \mathbf{y}^{(i-1)}(1) & \dots & \mathbf{y}^{(i-1)}(N-1) \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{7}$$

We then have:

$$\mathbf{P}^i = \mathbf{A}^i \cdot \mathbf{Q}^i$$

Applying transpose will lead to a more conventional system of linear equations with the unknown matrix $\mathbf{A}^{i,T}$ (secret weights and biases) at the rightmost position, which we call the *intra-layer linear system* for layer i:

$$\mathbf{P}^{i,T} = \mathbf{Q}^{i,T} \cdot \mathbf{A}^{i,T} \tag{8}$$

Since we can access both $\mathbf{y}^{(i-1)}$ and $\mathbf{y}^i$ using scan chains (define $\mathbf{y}^0 = \mathbf{x}^0$), using Equations 5 and 7 we can directly compute both $\mathbf{P}^{i,T}$ and $\mathbf{Q}^{i,T}$, $\forall i$, using N queries. The shape of the unknown parameter matrix $\mathbf{A}^{i,T}$ is $(a_{(i-1)}+1) \times a_i$. This aligns with the fact that there are a_i neurons in layer i, and each of them has $a_{(i-1)}$ weights along with 1 bias to solve. For each neuron j, Equation 8 transforms to $\mathbf{P}^{i,T}|_{column\ j} = \mathbf{Q}^{i,T} \cdot \mathbf{A}^{i,T}|_{column\ j}$, where $\mathbf{Q}^{i,T}$ matrix is reused $\forall j$.

Furthermore, input queries can flow unimpededly along with hidden layers. No matter what layer the adversary is intentionally extracting, the output of each hidden layer can all be recorded during the scan shift phase, avoiding additional and repetitive queries for different $\mathbf{Q}^i$s. This recycling feature of our scheme greatly improves scalability, as it greatly increases the utilization of each query and can thus save a large number of queries if the network is deep. On the other hand, the method in [10], having almost no query reuse, requires new queries for every incoming neuron.

The shape of $\mathbf{Q}^{i,T}$ is $N \times (a_{(i-1)}+1)$, hence,

$$rank(\mathbf{Q}^{i,T}) \leq \min\{N, a_{(i-1)}+1\} \tag{9}$$

(a) If $rank(\mathbf{Q}^{i,T}) = a_{(i-1)}+1$, Equation 8 becomes a *trivial least-squares problem*. Every neuron gets a unique solution of its parameters, and these extracted parameters exhibit *high fidelity* towards the original network. (b) However, if $rank(\mathbf{Q}^{i,T}) < a_{(i-1)}+1$, there are multiple solutions that satisfy Equation 8, becoming an *under-determined least-squares problem*. We show in the later sections that for layer i, it is not possible to achieve *high fidelity*, although high accuracy can still be retained.

TABLE I: Rank-deficiency of $\mathbf{Q}^i$ matrices in deeper networks and its impact on i^{th} layer's intra-layer linear system.

(a) $\mathbf{Q}^i$'s in 784-256-64-32-10 architecture, $i = 1,2,3,4$

Matrix	Dimension	Rank	$a_{(i-1)}+1$	Under-determined?
$\mathbf{Q}^1$	785×785	785	785	No
$\mathbf{Q}^2$	257×785	257	257	No
$\mathbf{Q}^3$	65×785	46	65	Yes
$\mathbf{Q}^4$	33×785	31	33	Yes

(b) $\mathbf{Q}^i$'s in 784-256-128-64-32-10 architecture, $i = 1,2,3,4,5$

Matrix	Dimension	Rank	$a_{(i-1)}+1$	Under-determined?
$\mathbf{Q}^1$	785×785	785	785	No
$\mathbf{Q}^2$	257×785	257	257	No
$\mathbf{Q}^3$	129×785	107	129	Yes
$\mathbf{Q}^4$	65×785	65	65	No
$\mathbf{Q}^5$	33×785	33	33	No

C. Query Number, Rank Deficiency, and Under-Determination

When the query number $N < a_{(i-1)}+1$, Equation 8 is clearly under-determined. But we observe from the experimental results later in Section V that even when N is astonishingly low, the extracted model's accuracy can still be reasonably high. This feature can act as a trade-off between the accuracy and the query count. When $N \geq \max\{a_0, a_1, \dots, a_{(n-1)}\}+1$, N is uncoupled from Inequality 9, and the inequality becomes $rank(\mathbf{Q}^{i,T}) \leq a_{(i-1)}+1$. In this way, whether $\mathbf{Q}^{i,T}$ is rank-deficient will be the sole determinant factor of whether system 8 is under-determined.

It is well-known that random matrices have full rank with a very high probability [15]. Additionally, we have observed empirically that when applied with a full-rank random matrix at the input of the neural network, due to the non-linear activation function, the input matrix to the subsequent layers has a high rank with a high probability. We exploit this observation to efficiently perform model extraction using uniformly distributed *random queries*.

As an example, Table I shows the ranks of the $\mathbf{Q}^i$ matrices in deeper network architectures, obtained empirically by simulating the network using random input queries. For both the architectures presented in Table I, we generated 785 uniformly distributed random input vectors to constitute the input matrix $\mathbf{Q}^1$. In this case, $N \geq \max\{a_0, a_1, \dots, a_n\}+1$, and $\mathbf{Q}^i$'s rank is the only factor that decides whether layer i sees an *under-determined least-squares problem* or just a *trivial least-squares problem*.

Due to heavy non-linearity, we observe that most of the neurons in the deeper layers produce outputs that are exceptionally close to either 0 or 1. These extreme values challenge the capacity of the hardware registers, also damaging the numerical diversity of $\mathbf{Q}^i (i \geq 2)$. As a result, some of the $\mathbf{Q}^i (i \geq 2)$ are degraded to rank-deficient matrices, as shown in Table I. It is worth noting that even if $\mathbf{Q}^{(i-1)}$ is rank-deficient, $\mathbf{Q}^i$ can still be full rank, thanks to the non-linear activation function. Nevertheless, as long as even

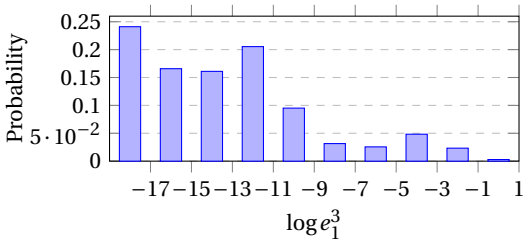


Fig. 3: Distribution of output errors for neuron 1 in layer 3.

one layer's *intra-layer linear system* is under-determined, the entire extracted network will lose its *high-fidelity* feature.

D. Under-Determined Least-Squares Regression

When layer i's *intra-layer linear system* (Equation 8) becomes under-determined, its parameters have multiple solutions. It is natural to desire the parameter solution $\mathbf{A}^i$ with the most generalization ability, that is, for any given input of layer i, this layer's output has the smallest deviation from the ground truth output. To achieve this goal, we find the $\mathbf{A}^i$ with the minimal $l2$ norm in the solution set.

$$\min \|\mathbf{A}^i\|_2, \quad s.t.\ \mathbf{Q}^{i,T} \cdot \mathbf{A}^{i,T} - \mathbf{P}^{i,T} = 0 \tag{10}$$

The importance of the minimal value can be explained by taking a look at the input matrix $\mathbf{Q}^i$'s distribution. As indicated earlier, normally the elements in $\mathbf{Q}^i$ for deeper layers are exceptionally close to either 0 or 1, as a consequence of heavy non-linearity. If the weights are not at their minimal value, the layer i will be more susceptible to arbitrarily small input values. If large weights are assigned to such small inputs, predictions on unseen data can be heavily biased by these weights if the corresponding inputs suddenly become close to 1 [16]. Therefore, to avoid this scenario, we require the weights to be at an overall minimum.

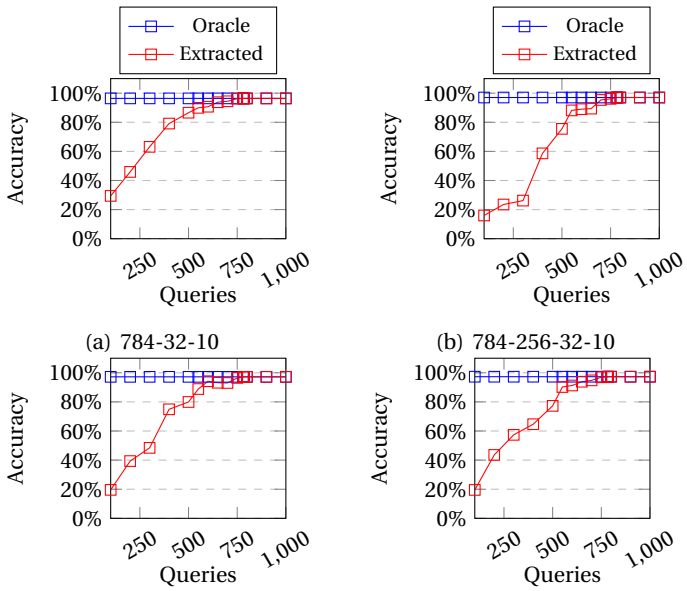
We refer to the ground truth NN model to be extracted as an *oracle*. In order to understand how similarly the extracted neurons in a certain layer function compared with the corresponding counterparts in the *oracle*, we introduce output error to quantify the deviation of an extracted neuron's output w.r.t. the corresponding neuron's output in the *oracle*:

$$e_j^i = \|y_{ex,j}^i - y_j^i\| \tag{11}$$

Equation 11 describes the Euclidean distance between the j^{th} neuron's output of the extracted model and the oracle in layer i. When the entire `MNIST` testing set (10000 images) is applied to these networks, 10000 entries of e_j^i will be produced. For instance, Figure 3 shows the general distribution of these errors when $i = 3, j = 1$. From the plot, it is evident that most of the errors are smaller than 10^{-9}. This indicates that the parameters obtained from under-determined `LSR` work substantially identical to the *oracle*'s neuron.

V. Experimental Evaluation

This section demonstrates the superiority of our `LSR` extraction method over state-of-the-art model extraction from ICCAD 2021 [10] under the same scan-chain threat model, by empirically demonstrating an improvement in extraction accuracy and with fewer queries. Further, we exploit the trade-off between input queries and the extracted model's accuracy and show that an accuracy of 90.22% can be reached with as few as 550 queries.



(a) 784-32-10 (b) 784-256-32-10

(c) 784-256-64-32-10 (d) 784-256-128-64-32-10

Fig. 4: Accuracy comparison between the oracle and the extracted model, for different FCNN topologies.

A. Experimental Setup

We have trained 14 different FCNN architectures using the `MNIST` digit recognition dataset [17], where 60000 images were used as the training set while 10000 images were used as the testing set. We adopted the *mean square error* (MSE) function as the loss function, with Adam [18] as the optimizer ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$). We have set the learning rate to 0.0025 and the batch size to 16.

We use *Python* to simulate our `LSR` attack. The random inputs are generated from the uniform distribution $U(0,d)$, $d < 1$, where d is a tunable parameter that varies across different networks, using `NumPy`. The ICCAD 2021 attack [10] is also implemented in *Python*, with the `SciPy` linear programming tool [19] as the constraint solver. All of our experiments are carried out on an Intel(R) Core(TM) i7-8750H Processor with 16 GB RAM.

B. Results

Table II shows the results obtained by implementing our `LSR` algorithm and comparing it with the prior work proposed in ICCAD 2021 [10], using 14 networks of different depths/sizes. This prior work adopts an *isolated activation* scheme for preciseness, requiring about $2(\sum_{i=0}^{n-1} a_i a_{i+1})$ queries for architecture a_0-a_1-a_2-...-a_n. In contrast, our algorithm `LSR` requires only $\max\{a_0, a_1, \ldots, a_{(n-1)}\}+1$ queries. In the case $\mathbf{Q}^i$s, $\forall i$, happen to be full rank, the exact *high-fidelity* solution can be obtained. However, even if not all $\mathbf{Q}^i$s are full rank, *high accuracy* for the extracted model can still be retained. The query counts can be *reduced further* at the cost of fidelity while achieving reasonable accuracy at the same time, as shown in Figure 4.

TABLE II: ICCAD 2021 corresponds to the high-fidelity scan-chain attack that requires *isolated activation* [10] and LSR corresponds to the proposed *approximate-fidelity* but high-accuracy *layer-constrained least-squares solution.*

Architecture	Original accuracy	ICCAD 2021 [10]		LSR (Proposed)			
		Queries	Accuracy	Queries	Reduction	Accuracy	Improvement
784-32-10	96.41%	51530	96.41%	785	66×	96.41%	0%
784-512-10	97.84%	824330	97.84%	785	1050×	97.84%	0%
784-1024-10	97.88%	1648650	97.88%	1025	1608×	97.88%	0%
784-1408-10	97.8%	2266890	97.76%	1409	1609×	97.8%	0.04%
784-2048-10	97.9%	3297290	97.9%	2049	1609×	97.9%	0%
784-256-32-10	97.01%	435922	96.89%†	785	555×	97.01%	0.12%
784-256-64-10	96.05%	468994	86.36%†	785	597×	96.05%	9.69%
784-512-64-10	97.06%	936890	95.93%†	785	1193×	96.1%	0.17%
784-512-128-10	97.36%	1066250	8.92%⋆	785	1358×	97.25%	88.33%
784-32-10-10-10	95.05%	52290	81.92%†	785	67×	95.05%	13.13%
784-32-32-32-10	96.03%	58186	11.21%⋆	785	74×	96.03%	84.82%
784-256-64-32-10	97.21%	471882	10.32%⋆	785	601×	97.21%	86.89%
784-32-10-10-10-10	94.68%	52350	10.1%⋆	785	66×	94.68%	84.58%
784-256-128-64-32-10	97.3%	561354	11.35%⋆	785	715×	97.3%	85.95%

† These accuracy degradations are caused by a small number of constraint solver failures.
⋆ When extracting these networks, the constraint solver fails an entire layer. Therefore, these accuracies are exceptionally low.

Due to the exceptionally large number of queries, the runtime of the prior work [10] is much greater than that of our extraction. The models extracted using our proposed method do not suffer any loss in accuracy, while most of the models extracted from [10] degrade in accuracy to various extents with increasing network depth.

Subsequently, we have explored the ability to trade off accuracy with queries in the context of our scalable least-squares extraction. We have demonstrated in Section IV-D that even when layer i's *intra-layer linear system* is under-determined, the solved layer exhibits equivalent external behavior. Inspired by this property, we explore the best accuracy the extracted model can achieve when the adversary has a tighter query budget (fewer than 785). We extract the model using queries ranging from 100 to 1000, and report the achieved classifier accuracy as shown in Figure 4.

We find that as queries increase beyond 100, the extracted model's accuracy increases rapidly. By the time the query counts reach 550, the extracted model's functional accuracy will have reached approximately 90% for most of the architectures. Subsequently, the rate of accuracy increase drops gradually until it reaches the oracle's accuracy.

VI. Discussion

In this section, we measure the efficacy of our attack, LSR, in the presence of existing countermeasures, and describe the limitations. FeatureSqueezing (FS) [12] and AdaptiveDenoising (AD) [13] are two input distortion defensive techniques primarily aiming against adversarial samples. FS [12] prunes the input space so that the adversarial noise gets eliminated while maintaining the major features of the normal inputs. FS consists of three components: bit-depth reduction (or quantization), local median smoothing, and non-local smoothing. AD [13] then further automated the denoising process in an adaptive manner.

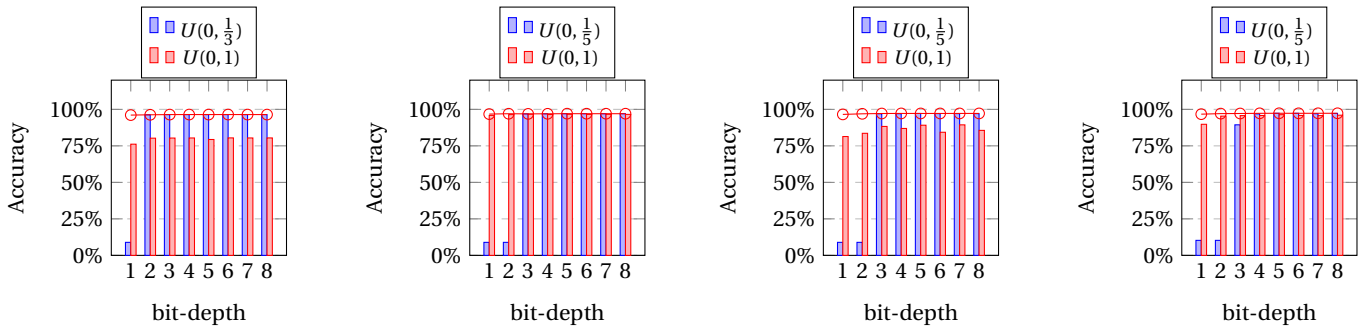
A. With *FeatureSqueezing*

Bit-depth reduction: [12] observed that quantizing the input to few grayscale levels through bit-depth reduction can cancel out low levels of adversarial noise. For example, if the input range is $(0,1)$, 1 bit-depth yields $2^{1-1}+1=2$ grayscale levels: $\{0,1\}$. 3 bit-depth yields $2^{3-1}+1=5$ grayscale levels: $\{0,0.25,0.5,0.75,1\}$. Every input value m is transformed to $\frac{[2^{n-1}\cdot m]}{2^{n-1}}$, where n is the bit-depth and $[\cdot]$ is the function for *rounding to the nearest integer.*

In practice, to cope with the severe non-linearity in deep networks, the adversary usually uses small input values (in $U(0,d),\ d<1$) so that $\mathbf{Q}^i$ is more likely to be full-rank, resulting in better accuracy. Since the input is small, the output of the first few layers will not reach deeply into the heavily non-linear regions. However, with the presence of bit-depth reduction, all the inputs will be rounded to 0 if the adversary decides to use smaller inputs. If the input distribution is $U(0,1)$, on the other hand, the bit-depth reduction will have less impact on the extraction, but the extracted model's overall accuracy may decrease.

As shown in Figure 5, if the bit-depth is larger, models extracted with input distribution $U(0,d)$ $(d<1)$ generally have better accuracy. However, if aggressively small bit-depth (e.g. 1 or 2) is used for better protection [12], all the inputs with $U(0,d)$ $(d<1)$ will be set to zero, while inputs with $U(0,1)$, however, remain valid, resulting in less accurate, yet valid model extractions. The adversary can then decide the value of d himself/herself based on the specific scenarios.

Local smoothing and non-local smoothing: *Median local smoothing* runs a sliding window over the pixels of the input image, and replaces the center pixel with the median of all the elements in the window. On the other hand, *non-local smoothing* finds similar patches in a larger portion of the input image and replaces the center patch with the average of all the similar patches. We have implemented *local smoothing* using `scipy.ndimage.median_filter` function from SciPy [20] and *non-local smoothing* using `cv2.fastNlMeansDenoisingColored` function from OpenCV [21] respectively. Smoothing has almost zero impact on our extraction, as shown in Table III.



(a) 784-32-10 (b) 784-256-32-10 (c) 784-256-64-32-10 (d) 784-256-128-64-32-10

Fig. 5: Impact on extracted model accuracy with bit-depth reduction squeezing. The red dots represent the oracles' accuracy.

TABLE III: Impact on accuracy with other input distortion methods. ⋆ refers to *local smoothing* method (with a median sliding window of (3,3)). † refers to *non-local smoothing* method (with a searching window of (13, 13), patch size is (3, 3) and filter strength is 2). We compare the accuracy of the *original* NN and the *extracted* NN.

Counter-measure	NN	784-32-10	784-256-32-10	784-256-64-32-10	784-256-128-64-32-10
LS⋆	ori.	95.03%	94.98%	95.82%	96.14%
	ext.	95.03%	94.98%	95.85%	96.24%
NLS†	ori.	94.3%	93.73%	94.58%	95.04%
	ext.	94.3%	93.73%	94.34%	94.61%
AD	ori.	94.15%	90.53%	94.31%	94.7%
	ext.	94.15%	90.53%	94.21%	94.5%

B. With `AdaptiveDenoising`

As mentioned, AD first calculates the input entropy and then chooses the denoising methodology. The input goes through a quantization process similar to bit-depth reduction and spatial filter smoothing. The difference, however, is that the parameters are not manually tuned but automatically adjusted based on entropy. The results of model extraction with AD are present in the last two rows of Table III.

C. Limitations

Although we have shown that our proposed LSR can extract large-scale deep neural networks with very few queries and achieve high accuracy, it has the following limitations.

Activation function: Our method has been tested on a bijective activation function. The adversary can exploit the bijective property and find the pre-activation value in each layer through scan chains to build intra-layer linear systems. Extension to non-bijective activation functions (e.g. ReLU) would be interesting future work.

Improvement in fidelity: Fidelity is not guaranteed in our approach, because uniformly distributed random inputs are not fully compatible with the heavy non-linearity of deeper networks. Due to this, it is common for $\mathbf{Q}^i$ for large i to be rank-deficient. Coupling input generation strategies with non-linearity would be interesting future work.

VII. Conclusions

We demonstrate a more powerful scan-chain attack compared to the state-of-the-art, which is capable of scaling to deeper/larger networks with *random queries* and *layer-constrained least-squares regression*. The proposed method shows ≈ 32% improvement in the classifier's accuracy on average, and multiple orders of reduction in queries, thereby demonstrating *scalability*. The proposed extraction methodology is also effective in the presence of existing countermeasures like FeatureSqueezing and AdaptiveDenoising. Further, we have demonstrated that it is possible to systematically trade-off queries with accuracy within our *least-squares* framework.

Acknowledgement

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust.

References

[1] F. Tramèr et al, "Stealing Machine Learning Models via Prediction APIs," in *USENIX Security Symposium*, 2016, pp. 601–618.

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

[3] M. Juuti et al, "PRADA: Protecting Against DNN Model Stealing Attacks," in *IEEE European Symposium on Security and Privacy*, 2019, pp. 512–527.

[4] D. Rolnick and K. P. Kording, "Reverse-engineering deep ReLU networks," in *ICML*, vol. 119, 2020, pp. 8178–8187.

[5] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic Extraction of Neural Network Models," in *CRYPTO*, vol. 12172, 2020, pp. 189–218.

[6] M. Jagielski et al, "High accuracy and high fidelity extraction of neural networks," in *USENIX Security Symposium*, 2020, pp. 1345–1362.

[7] D. Lowd and C. Meek, "Adversarial learning," in *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005, pp. 641–647.

[8] N. Papernot et al, "Practical black-box attacks against machine learning," in *AsiaCCS*, 2017, pp. 506–519.

[9] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[10] S. Potluri and A. Aysu, "Stealing neural network models through the scan chain: A new threat for ML hardware," *ICCAD*, pp. 1–8, 2021.

[11] "1149.1-2013 - IEEE Std. for Test Access Port and Boundary-Scan," 2013. [Online]. Available: https://standards.ieee.org/standard/1149_1-2013.html

[12] W. Xu et al, "Feature Squeezing: Detecting adversarial examples in deep neural networks," in *Network and Distributed System Security Symposium*, 2018.

[13] B. Liang et al, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 72–85, 2021.

[14] "ASE Group Test Service," 2021. [Online]. Available: https://ase.aseglobal.com/en/products/test

[15] X. Feng and Z. Zhang, "The rank of a random matrix," *Applied Mathematics and Computation*, vol. 185, no. 1, pp. 689–694, 2007.

[16] K. He, "SVD in machine learning: Underdetermined least squares," The University of Chicago, Tech. Rep., 2019.

[17] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[18] D. P. Kingma et al, "Adam: A method for stochastic optimization," *ArXiv*, 2014.

[19] "scipy.optimize.linprog," 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html

[20] "sciy.ndimage.median_filter," 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html

[21] "Denoising: Computational Photography," 2022. [Online]. Available: https://docs.opencv.org/4.x/d1/d79/group__photo__denoise.html