Juuti, M.; Szyller, S.; Marchal, S.; Asokan, N.

PRADA: Protecting Against DNN Model Stealing Attacks

PRADA: Protecting Against DNN Model Stealing Attacks

Mika Juuti, Sebastian Szyller, Samuel Marchal, N. Asokan

Aalto University

{mika.juuti, sebastian.szyller, samuel.marchal}@aalto.fi, asokan@acm.org

Abstract—**Machine learning (ML) applications are increasingly prevalent. Protecting the confidentiality of ML models becomes paramount for two reasons: (a) a model can be a business advantage to its owner, and (b) an adversary may use a stolen model to find *transferable adversarial examples* that can evade classification by the original model. Access to the model can be restricted to be only via well-defined prediction APIs. Nevertheless, prediction APIs still provide enough information to allow an adversary to mount *model extraction* attacks by sending repeated queries via the prediction API.**

In this paper, we describe new model extraction attacks using novel approaches for generating synthetic queries, and optimizing training hyperparameters. Our attacks outperform state-of-the-art model extraction in terms of transferability of both targeted and non-targeted adversarial examples (up to +29-44 percentage points, pp), and prediction accuracy (up to +46 pp) on two datasets. We provide take-aways on how to perform effective model extraction attacks.

We then propose PRADA, the first step towards generic and effective detection of DNN model extraction attacks. It analyzes the distribution of consecutive API queries and raises an alarm when this distribution deviates from benign behavior. We show that PRADA can detect all prior model extraction attacks with no false positives.

I. Introduction

Recent advances in deep neural networks (DNN) have drastically improved the performance and reliability of machine learning (ML)-based decision making. New business models like *Machine-Learning-as-a-Service* (MLaaS) have emerged where the model itself is hosted in a secure cloud service, allowing clients to query the model via a cloud-based prediction API. ML models are also increasingly deployed on end-user devices, and can similarly be deployed behind APIs using hardware security mechanisms. Model owners can monetize their models by, e.g., having clients pay to use the prediction API. In these settings, the ML model represents business value underscoring the need to keep it confidential.

Increasing adoption of ML in various applications is also accompanied by an increase in attacks targeting ML-based systems a.k.a. *adversarial machine learning* [39]. One such attack is forging *adversarial examples*, which are samples specifically crafted to deceive a target ML model [16]. To date, there are no effective defenses protecting against all such attacks [4] but one partial mitigation is to protect the confidentiality of the ML model.

However, prediction APIs necessarily leak information. This leakage of information is exploited by *model extraction attacks* [38], [55] where the adversary only has access to the prediction API of a target model which it can use as an oracle for returning predictions for the samples it submits. The adversary queries the target model iteratively using samples that are specifically crafted to maximize the extraction of information about the model internals via predictions returned by the model. The adversary uses this information to gradually train a *substitute model*. The substitute model itself may be used in constructing future queries whose responses are used to further refine the substitute model. The goal of the adversary is to use the substitute model to (a) obtain predictions in the future, bypassing the original model, and thus depriving its owner of their business advantage, and/or (b) construct *transferable adversarial examples* [49] that it can later use to deceive the original model into making incorrect predictions. The success of the adversary can thus be measured in terms of (a) prediction accuracy of the substitute model, and (b) transferability of adversarial samples obtained from the substitute model.

Prior extraction attacks are either narrowly scoped [38] (targeting transferability of a specific type of adversarial examples), or have been demonstrated only on simple models [55]. We are not aware of any prior work describing effective generic techniques to detect/prevent DNN model extraction.

Goal and contributions. Our goal is twofold. (1) demonstrate the *feasibility of model extraction attack on DNN models* by proposing new, more effective attacks, and (2) develop an effective generic defense to model extraction. By "generic", we mean applicability to models with any type of input data and any learning algorithm. We claim the following contributions:

- **novel model extraction attacks** (Sect. III), which, unlike previous proposals, leverage the optimization of training hyperparameters and generalize synthetic data generation approaches. They outperform prior attacks in transferability of targeted and non-targeted adversarial examples (+29-44 pp) and prediction accuracy (up to +46 pp) (Sect. IV-E).
- **new insights on model extraction success factors** showing that (a) cross-validated hyperparameter search outperforms selection of training hyperparameters (Sect. IV-B), (b) prediction probabilities help improve transferability of adversarial examples, while class labels are sufficient for high prediction accuracy for the substitute model (Sect. IV-C – IV-E), and (c) using the same architecture for the substitute model results in better transferablity while a more complex architecture can increase prediction

accuracy (Sect. IV-F).

- **a new technique, PRADA, to detect model extraction** which analyses the distribution of successive queries from a client and identifies deviations from a normal (Gaussian) distribution (Sect. V-A). We show that it is *effective*: 100% detection rate and no false positives on all prior model extraction attacks (Sect. V-B). To the best of our knowledge PRADA is the *first generic technique* for detecting model extraction.

We share the source code for our attacks on request for research use. Our defense is available as open source[1].

II. BACKGROUND

A. Deep Neural Network (DNN)

A deep neural network (DNN) is a function $F(x)$ producing output $y \in \mathbb{R}^m$ on input $x \in \mathbb{R}^n$, where $F(x)$ is a hierarchical composition of k parametric functions f_i ($i \in \{1, k\}$), each of which is a layer of neurons that apply *activation functions* to the weighted output of the previous layer f_{i-1}. Each layer is parametrized by a weight matrix θ_i, a bias b_i and an activation function σ_i: $f_i(x) = \sigma_i(\theta_i \cdot x + b_i)$. Consequently a DNN can be formulated as follows:

$$F(x) = f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1 \circ x \qquad (1)$$

$$F(x) = \sigma_k(\theta_k \cdot \sigma_{k-1}(\theta_{k-1} \cdots \sigma_1(\theta_1 \cdot x + b_1) \cdots + b_{k-1}) + b_k) \qquad (2)$$

In this paper we focus on predictive DNNs used as m-class classifiers. The output of $F(x)$ is an m-dimensional vector containing the probabilities p_j that x belongs to each class c_j for $j \in \{1, m\}$. The last activation function σ_k is typically *softmax*. A final prediction class[2] $\hat{F}$ is obtained by applying the *argmax* function: $\hat{F}(x) = argmax(F(x)) = c$.

B. Adversary Description

The adversary's objective is to "steal" a target machine learning model F by making a series of prediction requests $U = \{x_1, \ldots, x_n\}$ to F. Responses $Y = \{\hat{F}(x_1), \ldots, \hat{F}(x_n)\}$ along with U, are used by adversary to train its substitute model F'. Model extraction attacks [38], [55] operate in a *black-box* setting. The adversary does not have access to all target model internals, but has access to a prediction API.

Model extraction to date operates in settings where the adversary does not have a large set of "natural" samples. These attacks require crafting and querying of *synthetic samples* to extract information from F. This attack pattern is increasingly more realistic given the emergence of the MLaaS paradigm, where one party (model provider) uses a private training set, domain expertise and computational power to train a model. The model is then made available to other parties (clients) via a prediction API on cloud-based platforms, e.g., AmazonML [1] and AzureML [32]. The models are monetized by charging fees from clients for each prediction made by the

[2]We use the hat notation ˆ to denote predictions

models. The model provider may alternatively deploy models on client devices (e.g. smartphones or cameras), and rely on platform security mechanisms on these devices, to protect model confidentiality. What is common to both scenarios is that while the models may be secured against physical theft, the prediction APIs will remain open, enabling model extraction attacks relying on predictions.

C. Goals

Adversaries are incentivized to extract models for (Sect. I):

- **Reproduction of predictive behavior.** The purpose of the substitute model F' is to reproduce as faithfully as possible the prediction of F for a known subspace S of the whole input space R^n, i.e. $\forall x \in S \subset \mathbb{R}^n$. It may be:
  - The whole space of input values $S = R^n$, in which case *all* predictions made by F' will match the predictions of F. This *Random Uniform Agreement* is measured by randomly sampling the input space.
  - A relevant subset of the whole input space, e.g. all images x that are in the subset "digits" when attacking a digit classifier. *Agreement* is measured by sampling a held-out test set that neither classifier has seen before.

- **Transfer of adversarial examples.** Forging adversarial examples consists of finding minimal modifications ϵ for an input x of class c such that $x' = x + \epsilon$ is classified as $c' \neq c$ by a model F. Adversarial examples are either:
  - *Targeted,* where $x + \epsilon$ is created to change the classification of x from c to *a specific class* c'.
  - *Non-targeted,* where $x + \epsilon$ is created to change the classification of x from c to *any other class* c'.

Secondarily, adversaries want to minimize the number of prediction queries to F in order to (1) avoid detection and prevention of the attack, (2) limit the amount of money spent for predictions, in the case of MLaaS prediction APIs, and to (3) minimize the number of natural samples required to query the model.

D. Adversary Model

Attack surface. We consider any scenario where the target model is isolated from clients by some means. This can be a *remote isolation* where the model is hosted on a server or *local isolation* on a personal device (e.g. smartphone) or an autonomous system (e.g., self-driving car, autonomous drone). We assume local and remote isolation provide same confidentiality guarantees and physical access does not help the adversary to overcome the isolation. Such guarantees can be typically enforced by hardware assisted TEEs [13]. Increasing availability of lightweight hardware enhanced for DNNs [22] and the rise of federated learning will push machine learning computation to the edge [24], [46]. Local isolation will become increasingly adopted to protect these models.

Capabilities. The adversary has black-box access to the isolated target model. It knows the shape of the input (n) and output (m) layers of the model. It knows the model architecture: intermediate layer shapes and activation function types.

It can query samples x to be processed by the model and gets the output predictions. Predictions may be *labels only* $\hat{F}(x)$, or full set of *probabilities* $F(x)$ which is a m-dimensional vector containing the probabilities p_i that x belongs to each class c_i for $i \in \{1, m\}$. Classes c_i are meaningful to the adversary, e.g., they correspond to digits, vehicles types, prescription drugs, etc. Thus, the adversary can assume what the input to the model looks like even though it may not know the exact distribution of the data used to train the target model[3].

E. General Model Extraction Process

We present a general process for extracting neural network models through prediction APIs. Assuming a target model F, we want to learn a substitute model F' to mimic behavior of F (Section II-C). The maximum number of queries may be limited to a query budget b. We detail this model extraction process in Algorithm 1 and discuss some of the steps next:

Initial data collection. The adversary composes an initial set U of unlabeled samples (*seed samples*, row 6). The source of these samples is determined by the adversary's capabilities. Typically knowledge of input shape is assumed. All samples are queried from F, and responses are collected into dataset $L = \{U, \hat{F}(U)\}$ (row 7).

Architecture and hyperparameters. The adversary selects a neural network architecture (row 8) and hyperparameters (row 9) for F'. F' is trained with L. After this, ρ duplication rounds (iterative steps) are run.

Duplication rounds. The adversary increases coverage of the input space by generating synthetic samples. This generation typically leverages knowledge of F acquired until then: labeled training samples L and current F'. This synthetic data is allocated to a new set U (row 13). All, or part, of the unlabeled synthetic samples $x \in U$ are queried from F, to get predictions $\hat{F}(x)$. These new labeled samples are added to L (row 14). Labeled samples L are used for training F' (row 15).

Duplication rounds are repeated until the prediction query budget b is consumed or termination occurs otherwise. The outcome is a substitute model F' that mimics behavior of F.

F. Prior Model Extraction Attacks

We present two main techniques that have been introduced to date for model extraction. These are used as a baseline to improve model extraction attacks and compare performance (Sect. IV) and to evaluate our detection approach (Sect. V).

1) TRAMER attack [55]: Tramer et al. introduced several attacks to extract simple ML models including the one-layer logistic regression model with an *equation solving* attack and decision trees with a *path finding* attack. Both have high efficiency and require a few prediction queries but are limited to the simple models mentioned. These attacks are specifically

---

Algorithm 1 Model extraction process with the goal of extracting classifier F, given initial unlabeled seed samples X and a substitute model F' (initially random).

1: **procedure** LABEL($\{x_1, \ldots, x_n\}, F$)
2:     **return** $\{\hat{F}(x_1), \ldots, \hat{F}(x_n)\}$ ▷ Return predictions
3: **end procedure**
4:
5: **procedure** EXTRACTMODEL(F)
6:     $U \leftarrow$ *Initial data collection*
7:     $L \leftarrow \{U, \text{LABEL}(U, F)\}$
8:     $F' \leftarrow$ *Select architecture*
9:     $H \leftarrow$ *Resolve hyperparameters*   ▷ cf. Sec. III-A
10:     $F' \leftarrow$ INITIALIZE(F')   ▷ Set random weights
11:     $F' \leftarrow$ TRAIN($F' \mid L, H$)
12:     **for** $i \leftarrow 1, \rho$ **do**   ▷ ρ duplication rounds
13:         $U \leftarrow$ *Create synthetic samples*   ▷ cf. Sec. III-C
14:         $L \leftarrow \{ L \cup \{U, \text{LABEL}(U, F)\} \}$
15:         $F' \leftarrow$ TRAIN($F' \mid L, H$)
16:     **end for**
17:     **return** F'
18: **end procedure**

---

designed to reach very high *Random Uniform Agreement* (Sect. II-C). The authors also introduce an extraction method targeting shallow neural networks that we present below according to our process (Sect. II-E).

Initial data consists of a set U of uniformly selected random points of the input space (row 6). No natural samples are used. The attack assumes knowledge of the model architecture, hyperparameters and training strategy used for F (rows 7–8).

The main contribution for extracting neural networks lies in the prediction queries (row 13), where they introduce three strategies for querying additional data points from F. The first selects these samples randomly. The second called *line-search retraining* selects new points closest to the decision boundary of the current F' using a line search technique. The last is *adaptive retraining* which has same intuition of querying samples close to the decision boundary, but it employs active learning techniques [9]. The first two techniques are implemented[4], and we evaluate the strictly stronger line-search retraining technique as TRAMER in this work. This technique initially queries 25% of the budget with random data (row 6), and then constructs line-search queries with 75% of remaining budget in one duplication round (row 13).

2) PAPERNOT attack [38]: Papernot et al. introduced a model extraction attack that is specifically designed at forging transferable *non-targeted* adversarial examples (Sect. II-C). We present this technique according to our process.

Initial data consists of a small set of natural samples (row 6). These are disjoint samples but distributed similarly as the target model's training data. Seed samples are balanced (same number of samples per class) and their required number increases with the model input dimensionality. The attack

---

[3]This is similar to [16] but differ from [55] which assumes that the adversary has no information about the purpose of classification, i.e, no intuition about what the input x must look like. We consider that classes must be meaningful to provide utility to a client and that the adversary has access to a few natural samples for each class.

[4]https://github.com/ftramer/Steal-ML

does not assume knowledge of F, hyperparameters or training strategy, however expert knowledge is used to select a model architecture "appropriate" for the classification task of F (row 8). Two strategies are proposed to query F (row 14), one queries the whole set U while the other called *reservoir sampling* queries a random subset of $X\%$ samples from U. Unselected samples are thrown away. PAPERNOT is defined with a fixed training strategy: Stochastic Gradient Descent [34] with learning rate 0.01, and momentum 0.9. F' is trained for a very short time (10 epochs) at each duplication round, to save time and to avoid overfitting L.

They introduce the *Jacobian-based Dataset Augmentation* (JbDA) technique for generating synthetic samples (row 13). It relies on computing the Jacobian matrices with the current F' evaluated on the already labeled samples in L. Each element $x \in L$ is modified by adding the sign of the Jacobian matrix $\nabla_x \mathcal{L}(F'(x, c_i))$ dimension corresponding to the label assigned to x by F, evaluated with regards to the classification loss $\mathcal{L}$. Thus, the set U is extended with $\{x + \lambda \cdot sign(\nabla_x \mathcal{L}(F'(x, c_i)))\}, \ \forall x \in L$. U has the same size as L, which means that the number of generated synthetic samples doubles at each iteration. λ is fixed to 25.5/255 in their evaluation. Thus, the creation of synthetic samples is identical to calculation of adversarial examples using the *Fast Gradient Sign Method* (FGSM) [16] on F', and augmenting the attacker's set L with their classification labels $\tilde{F}(x)$. The duplication rounds are repeated for a predefined number of ρ iterations, which they call *substitute training epochs*.

III. DNN MODEL EXTRACTION FRAMEWORK

Techniques proposed to date [55], [38] are narrowly scoped and explored solutions for only some of the required steps (Sect. II-C). We investigate several strategies for two crucial steps of the model extraction process: *selecting hyperparameters* (Algorithm 1, row 9) and *synthetic sample generation* (row 13), and investigate what advantage probabilities (rather than label responses) give to the adversary (rows 7 and 14). In addition, in Sect. IV-C we explore the impact of natural sample availability during initial data collection (row 6), and in Sect. IV-F what impact mismatch in model architectures have on attack performance (row 8).

A. Hyperparameters

Predictive performance of neural networks is highly dependent on hyperparameters used for training. These include the *learning rate*, and the *number of training epochs*. Too low a learning rate may preclude finding the optimal solution before termination whereas too high a rate can overshoot optimal solutions. There are essentially three ways of choosing hyperparameters in model extraction attacks:

- *Rule-of-thumb.* Use some heuristic. E.g. PAPERNOT [38] uses a fixed learning rate and small number of epochs.
- SAME. Copy from the target model. This may be obtained via insider knowledge, or through state-of-the-art attacks [36].

Algorithm 2 Five-fold cross-validation (CV) search using bayesian optimization, given labeled dataset L, and closed linear span of $\mathcal{H}$ (hyperparameters range: [learning rate] $\times$ [train epochs]). The procedure searches for the best hyperparameter combination H_{i*} that maximizes 5-fold CV accuracy.

1: **procedure** SAMPLE($L_{\text{train}}, L_{\text{val}}, H$) $\triangleright$ Calc. CV-accuracy
2:      $F' \leftarrow$ INITIALIZE(F') $\triangleright$ Set random weights
3:      $F' \leftarrow$ TRAIN($F' \mid L_{\text{train}}, H$)
4:      $accuracy \leftarrow$ EVALUATE(F', L_{val})
5:      **return** $accuracy$
6: **end procedure**
7:
8: **procedure** 5-FOLDSAMPLE(H) $\triangleright$ Average over 5 folds
9:      **for** $i \leftarrow 1, 5$ **do**
10:          $acc_i \leftarrow$ SAMPLE($L_{\text{train}}^i, L_{\text{val}}^i, H$)
11:      **end for**
12:      **return** MEAN($acc_1, \dots, acc_5$)
13: **end procedure**
14:
15: **procedure** CV-SEARCH($F', L, \mathcal{H}$)
16:      $(L_{\text{train}}^1, L_{\text{val}}^1), \dots, (L_{\text{train}}^5, L_{\text{val}}^5) \leftarrow$ KFOLDS($L, 5$)
17:      **for** $i \leftarrow 1, 4$ **do** $\triangleright$ Sample each corner of $\mathcal{H}$
18:          $H_i \leftarrow GetVertex(\mathcal{H}, i)$
19:          $y_i \leftarrow$ 5-FOLDSAMPLE(H_i)
20:      **end for**
21:      **for** $i \leftarrow 5, 15$ **do** $\triangleright$ Sample randomly inside $\mathcal{H}$
22:          $H_i \leftarrow UniformRandom(\mathcal{H})$
23:          $y_i \leftarrow$ 5-FOLDSAMPLE(H_i)
24:      **end for**
25:      **for** $i \leftarrow 16, 30$ **do** $\triangleright$ Sample with Gauss. Process *GP*
26:          $GP \leftarrow$ INITIALIZE() $\triangleright$ Set random weights
27:          $GP \leftarrow$ *Train GP to predict* $y_{1,\dots,i-1}$ *from* $H_{1,\dots,i-1}$
28:          $H_i \leftarrow$ *Find next value that GP perceives maximizes* "*expected value + standard deviation*"
29:          $y_i \leftarrow$ 5-FOLDSAMPLE(H_i)
30:      **end for**
31:      $i^* \leftarrow \arg\max(y_i)$
32:      **return** H_{i^*}
33: **end procedure**

- CV-SEARCH. Do a *cross-validation search* on the initial seed samples (row 6).

In this paper, we conduct CV-SEARCH by five-fold cross-validation. Five-fold cross-validation proceeds as follows. For each hyperparameter combination we want to test out, the initial labeled dataset L (row 7) is divided into 5 non-overlapping sets. The average accuracy is aggregated over the sets, is saved, and next hyperparameter combination is tested out. The process is repeated 5 times, each with a different validation set. The hyperparameter combination that produces the best accuracy on validation sets is selected for the rest of the attack.

Given a finite time, not all parameter combinations can be tested out. While strategies like *grid search* and *random search*

[15] are popular, Bayesian hyperparameter optimization [47] is more efficient: after first querying some initial samples, it estimates what validation accuracy certain hyperparameter combinations might have, along with the uncertainty of these estimates. Then the next test hyperparameter combinations are chosen as the ones that have either high expected value or high uncertainty[5]. CV-SEARCH is done with dropout training [34]. We detail our CV-SEARCH procedure using Bayesian Optimization in Algorithm 2. Learning rate is searched between 10^{-4} and 10^{-2}, and training epochs between 10 and 320. Both are searched in log-scale.

B. Adversarial Example Crafting

Adversarial examples are crafted by modifying samples $x \in R^n$ with the Jacobian matrix for a given DNN F with C classes, which in turn tells what the impact of each feature is on the overall classification loss $\mathcal{L}$ [34].

The Jacobian is used for finding out *how* to modify the features of a sample x such that the sample is classified as something different from its genuine class c_i. To modify x into a *targeted* adversarial example of class $c_j \neq c_i$, the Jacobian component on column j is used, such that x is modified in the *negative* gradient direction $x' \leftarrow x - f(\nabla_x \mathcal{L}(F(x, c_j)))$ with some function f.

To create a *non-targeted* adversarial example, the ith column of the Jacobian is used. The sample x is modified in the general *positive* gradient direction, to *decrease* the likelihood of classifying it as a member of class c_i: $x' \leftarrow x + f(\nabla_x \mathcal{L}(F(x, c_i)))$ For brevity we only discuss the non-targeted variant here.

The form of f determines the adversarial example crafting algorithm. Popular choices are Fast Gradient Sign Method *FGSM* [16], and its iterative variants *I-FGSM* [26] and *MI-FGSM* [12]. The overall modification for each of these algorithms is bounded to remain within an L_∞ distance of ϵ.

a) FGSM: A sample x of class c is modified by the *sign*-function of the gradient and multiplied by a small ϵ,

$$x' \leftarrow x + \epsilon \cdot sign(\nabla_x \mathcal{L}(F(x, c_i))) \qquad (3)$$

FGSM is called a "one-step method", and until recently, it was thought that these methods are most effective at producing transferable adversarial examples [12].

b) I-FGSM: Iterative FGSM subdivides modifications into k steps, such that every iterative modification is done with FGSM with step size $\frac{\epsilon}{k}$.

c) MI-FGSM: Momentum Iterative FGSM was recently shown to be the strongest method of creating transferable adversarial examples when attacking DNN models [12]. MI-FGSM includes a momentum term that accumulates previous gradient directions [12]. MI-FGSM won both the targeted and non-targeted adversarial example challenge at NIPS 2017 Adversarial Attack competition.
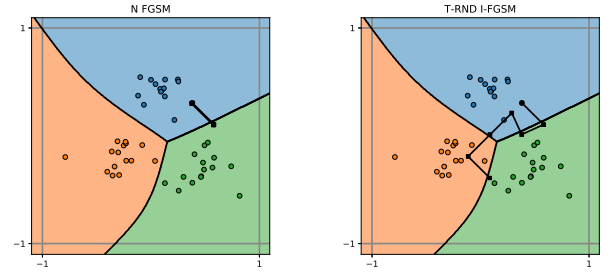
(a) Non-targeted FGSM   (b) T-RND I-FGSM

Fig. 1: Synthetic sample generation against a multi-layer perceptron. We show *six sequential steps*. Left: the non-targeted FGSM [38] does not generate novel data points after the first step. Right: T-RND I-FGSM avoids this by varying the contribution of features, and targeting random classes.

C. Synthetic Sample Generation

Synthetic samples in model extraction attacks can be constructed either using the partially trained substitute model F', or independently of it. We call these strategies *Jacobian-based Synthetic Sample Generation* and *Random Synthetic Sample Generation* respectively.

We create new synthetic samples with regards to all previously labeled data: the number of new samples increases exponentially with the number of duplication rounds ρ (Algorithm 1, row 12). We call the rate at which the number of synthetic samples grows the *expansion factor k*.

1) Jacobian-based Synthetic Sample Generation: These variants use adversarial example crafting algorithms (Sect. III-B) to produce new synthetic samples. Previous work [38] considered using non-targeted FGSM. We consider several choices, particularly targeted variants. All variants produce synthetic samples that step closer and closer to the perceived classification boundaries over the course of several duplication rounds (Algorithm 1, row 12).

We illustrate the intuition for the effect different algorithms have in Fig. 1. For this, we trained a multi-layer perceptron (MLP) [34] over two-dimensional toy data with three classes. We show six steps, which corresponds to six duplication rounds in [38]. We first demonstrate the synthetic sample crafting method of using <u>N</u>on-targeted <u>FGSM</u> [38]. Non-targeted variants try to greedily move towards the closest other class. If the classifier is not updated sufficiently between runs, the algorithm behaves like in Fig. 1a, where synthetic samples start to overlap, and do not contribute with new information about the target model F. The overlapping behavior can be avoided to a certain degree by stepping in a *targeted* <u>rand</u>omly chosen direction (T-RND), as in Figure 1b. Importantly, overlap can be further avoided by using <u>i</u>terative <u>FGSM</u> methods (I-FGSM) that can vary the contribution of different feature components.

For non-targeted methods, the expansion factor is always $k = 2$. However, for targeted variants, k can be as high as the number of classes C. We set $k = 4$ for the targeted variants

in our tests.

2) Random Synthetic Sample Generation: In addition to these, we consider a generic synthetic sample generation method: randomly perturbing <u>color</u> channels (COLOR). For grayscale images, COLOR randomly increases or decreases luminosity by a step size λ. For colored images, COLOR randomly perturbs the color channel of each pixel by the same amount for a given color channel. Random synthetic sample generation methods can have arbitrary expansion factors, but we set $k = 4$ in this paper.

IV. DNN Model Extraction: Evaluation

In this section, we replicate prior techniques for model extraction [55], [38], to explore the effect of different parameter choices and develop new, more effective model extraction attacks.

A. Experiment Setup

a) Datasets and target model description: We evaluate two datasets: MNIST [28] for digit recognition and GT-SRB [50] for traffic sign recognition. We chose these datasets because they had been evaluated in previous studies [38], and we wish to validate their observations under our adversary model (Sect. II-C – II-D). MNIST contains 70,000 images of 28×28 grayscale digits (10 classes). Its training set contains 60,000 and the rest are in the test set. GTSRB contains 39,209 images in the training set, and 12,630 images in the test set (43 classes). Images in GTSRB have different shapes (15×15 to 215×215); we normalize them to 32×32. We additionally scale feature values for both datasets to the range [-1, 1].

MNIST	GTSRB
conv2-32	conv2-64
maxpool2	maxpool2
conv2-64	conv2-64
maxpool2	maxpool2
FC-200	FC-200
FC-10	FC-100
	FC-43

TABLE I: Target models architecture (ReLU activation between blocks).

We use the model architectures depicted in Tab. I for training our target models. At a high level, we separate three disjoint sets of data for our experiments: *test* set, *target model training* set, and *pre-attacker* set. We call the set of initial seed samples in the model extraction process (Algorithm 1, row 6) the *attacker* set, and it is a subset of *pre-attacker* set. We vary its size systematically to understand the dependence of model extraction performance metrics on initial seed samples (Sect. IV-C for details).

In MNIST we train 10 target models. The *target model training* sets and *pre-attacker* set are obtained by first separating the "training" set of MNIST with stratified 10-fold cross-validation, giving approximately a 6,000:54,000 split. The larger of these sets is used for *target model training*. In GTSRB, we separated 36,629 images for *target model training* and 2,580 for *pre-attacker* set. GTSRB consists of up to 30 non-iid sequential samples of same physical objects photographed at different distances and angles. We ensured that same physical objects were only present in one of the datasets. We reserve the 12,630 test images for *test* set. We trained all target models for 100 epochs using Adam [15] with learning rate 0.001. The learning rate was halved when the cost plateaud. The models reached on average 98% accuracy on MNIST test set, and 95% on GTSRB test set.

b) Technicalities: We measure the reproduction of predictive behavior with the *agreement* metrics. It represents the accuracy of the substitute model predictions when taking the target model prediction as ground truth, i.e., a count of $\hat{F}'(x) = \hat{F}(x)$ occurrences. We compute *Test-agreement* for a relevant subset of the input space as a macro-averaged F-score using MNIST and GTSRB test sets. This metric faithfully reports the effectiveness of an attack even in the case that classes are imbalanced. We compute the random uniform agreement *RU-agreement* as an accuracy score on 4,000 samples chosen uniformly at random in the input space.

We measure *transferability* of adversarial examples over all seed samples in the attacker set. We measured both *Targeted* and *Non-targeted* transferability. We use the maximum perturbation $\epsilon = 64/255$ in our attack. For *Targeted*, we create 9 variants x' of the initial sample x with $\hat{F}(x) = c$, targeting 9 different classes $c' \neq c$.

Adversarial examples can be crafted with a variety of maximum perturbations ϵ. Larger values increase transferability, while impacting the visual perception of images to humans [45]. The end-goal of our paper is not to discuss how these choices impact human perception. Creation of transferable adversarial examples simply serves as a way to evaluate the success of model extraction attacks. For this reason, we choose the middle-range value of $\epsilon = 64/255$ throughout our paper. This value was the middlemost value evaluated in Papernot et al. [38].

Our settings differ from prior works as follows: Tramer et al. [55] did not evaluate model extraction attacks on DNNs; their largest neural network had $2,225$ parameters, while our smallest network (MNIST) has $486,011$ parameters. The datasets they evaluated were smaller than the ones we use; our datasets are the same as in Papernot et al [38]. Papernot et al. evaluated their attack up until $6,400$ queries, while we increase the total number of queries to $102,400$. *Test-agreement* in both earlier works is estimated with accuracy, while we use macro-averaged F-score to faithfully report agreement for the underrepresented classes in the datasets, which is important in GTSRB. Finally, for GTSRB we ensured that the test set contained different physical images compared to the attacker set, whereas Papernot et al. did not.

B. Evaluation of prior attacks and hyperparameters

We first evaluate PAPERNOT and TRAMER (Sect. II-F). For brevity, we only report results on MNIST although the results for GTSRB showed similar results. We report results for PAPERNOT and TRAMER on GTSRB further up in Sect. IV-E.

For PAPERNOT, we use 10 natural seed samples per class (100 samples), and end after 10 duplication rounds ($2^{10} \cdot 100 = 102,400$ samples in total). TRAMER initially queries random samples, followed by a line-search to find synthetic

(a) *Test-agreement*. (b) *RU-agreement*. (c) *Non-targeted* transferability. (d) *Targeted* transferability.
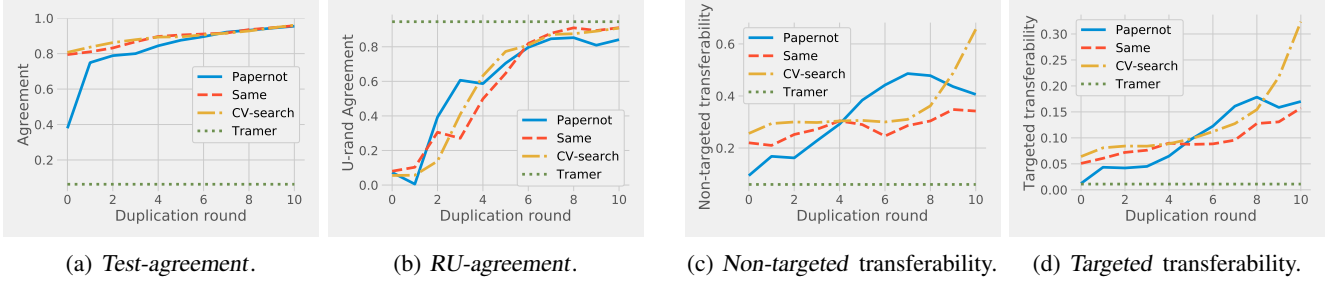
Fig. 2: Model extraction performance vs. duplication rounds for four attack setups (Sect. IV-B) on MNIST. Mean results over 10 independent attacker sets. Transferability is significantly improved by using CV-SEARCH strategy both without synthetic data augmentation (duplication round 0), and after using 102,300 synthetic samples (duplication round 10).

(a) MNIST *Test-agreement*. (b) MNIST *Targeted*. (c) GTSRB *Test-agreement*. (d) GTSRB *Targeted*.
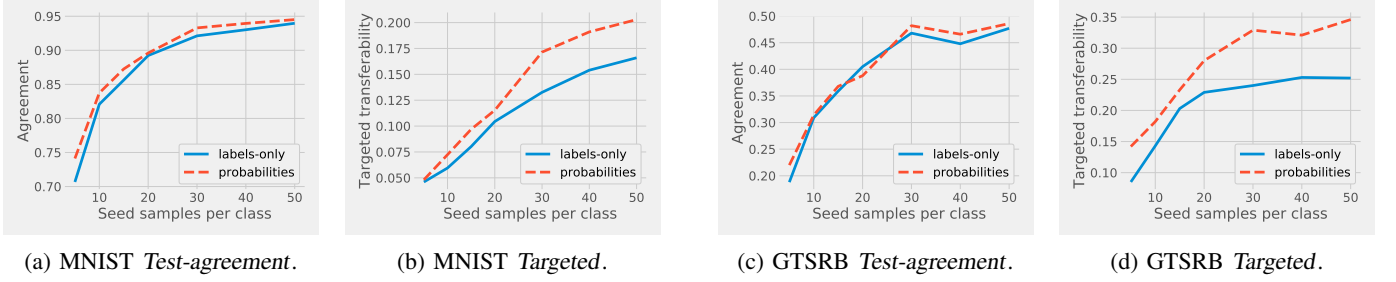
Fig. 3: Effect of number of seed samples on model extraction performance on MNIST and GTSRB. No synthetic samples are queried at this phase. The substitute model is trained with the CV-SEARCH strategy.

samples lying between existing ones, for a total of $102,400$ synthetic queries. We report the stronger TRAMER, which uses probability outputs. Notably, TRAMER uses 100% synthetic samples, whereas PAPERNOT use 0.1% natural samples and 99.9% synthetic samples. In addition to these, we test two new variants of PAPERNOT with new hyperparameters (Sect. III-A).

At this state, we only vary the hyperparameter setup. We show the evolution of agreement and transferability as the number of duplication rounds increases in Fig. 2. TRAMER only uses one duplication round, so we show it as a straight line for clarity. Transferability is computed using MI-FGSM with step size $\epsilon = 64/255$. There are several interesting observations: 1) TRAMER produces the best *RU-agreement*, but the performance translates neither to good *Test-agreement*, nor to good transferability. 2) both initial agreement and transferability are highest for models trained with CV-SEARCH, even higher than on SAME. 3) *Test-agreement* on MNIST is ultimately dominated by the high volume of synthetic samples in the attacker set: all training setups converge to the same *Test-agreement*. 4) *RU-agreement* is initially random (Fig. 2) – the accuracy of these is approximately $\frac{1}{C} = 10\%$, where C is the number of classes. The more synthetic queries are sampled, the higher the *RU-agreement* rises. However, the increase stagnates at the sixth duplication round (at 6400 samples), and does not rise further for any method. We suspect that this is due to the limitations of FGSM (recall discussion regarding Fig. 1 in Sect. III-C). 5) CV-SEARCH is the only method where transferability starts improving exponentially after the seventh duplication epoch. We verified that this effect is due to *dropout*

training. Other models may not improve due to overfitting model parameters on the substantial *attacker* set data, and dropout may help in avoiding this phenomena. 6) PAPERNOT is the fastest attack among these: querying and training (without network latency) took on average 4.5 minutes, while it took 26 min and 18 min respectively for the CV-SEARCH and SAME attacks.

We report detailed transferability for PAPERNOT in Tab. II after the last duplication round, calculated with different adversarial example crafting algorithms (Sect. III-B), including the FGSM evaluated in [38]. The iterative algorithms are run with 11 steps each. We also show a *random* perturbation of the same size for comparison. To be brief, we only report transferability for PAPERNOT, although the results for the different attacks showed the same pattern. The actual numbers for transferability differs from [38], as the target model architecture they attacked is not disclosed.

Evasion method	*Non-targeted* (%)	*Targeted* (%)	L_2 norm
random	7.5 ± 2.7	1.0 ± 0.4	10.3 ± 0.01
FGSM	24.4 ± 5.9	8.3 ± 2.0	9.2 ± 0.04
I-FGSM	57.4 ± 7.1	12.2 ± 2.3	9.3 ± 0.04
MI-FGSM	40.6 ± 8.4	17.0 ± 4.3	8.5 ± 0.05

TABLE II: MNIST. Transferability of PAPERNOT after 10 duplication rounds, evaluated with different evasion attacks. We use L_∞ bound $\epsilon = 64/255$ for transferability calculation. L_2 norm is shown for *Non-targeted*, for all generated samples. Mean and standard deviations shown.

We find that although PAPERNOT uses FGSM to craft

synthetic samples, transferability is better when crafting adversarial examples using the iterative variants of the algorithm: I-FGSM and MI-FGSM. We can see that iterative variants of the attack are stronger in our scenario. After synthetic queries had been queries, we observed that *Non-targeted* is $2\times$, and *Targeted* up to $3\times$ higher for iterative variants of the attack. For this reason, transferability will only be evaluated using iterative variants of the attack in the rest of this paper.

C. Impact of seed samples

Next, we explore the connection between the number of seed samples and model extraction efficiency, to understand the impact that these have on overall attack efficiency. We use the CV-SEARCH training strategy, which we demonstrated worked the best, and do not query synthetic data at this stage. We also investigate whether more detailed information from the target models – the full list of classification *probabilities* rather than the top-1 *labels-only* – can aid in the model extraction. These may be considered the highest and respectively lowest levels of granularity that any prediction API may provide.

Figure 3 shows *Test-agreement*, and *Targeted* for MNIST and GTSRB, as the number of training samples (natural seed samples) increases. Transferability is calculated with MI-FGSM. Overall *Test-agreement* trends are similar on both datasets, but *Test-agreement* is smaller on GTSRB than on MNIST. We believe this is due to higher dissimilarity between samples in attacker set, and target model training set.

Test-agreement does not improve significantly with probabilities; we observe increases between 0 and 3 percentage points (pp). This is in contrast with findings on shallow architectures in [55], where probabilities increased model extraction efficiency significantly. Increasing the number of training samples to 10-fold (from 5 samples per class to 50 samples per class) increases *Test-agreement* by 23 pp on MNIST and 29 pp on GTSRB, reaching 93% and 47% *Test-agreement* respectively.

Targeted improves with probabilities. The effect is more pronounced, when the adversary has access to more seed samples. For MNIST, *Targeted* starts at 5% when there are only 5 seed samples per class, and reaches 16% with 50 seed samples per class with labels-only, and 20% with probabilities. Perhaps surprisingly, *Targeted* on GTSRB is higher than on MNIST. *Targeted* stagnates in GTSRB after $20 - 30$ seed samples per class. We believe that the stagnation occurs due to correlated samples in the seed sample set, due to the structure of GTSRB dataset. When 50 samples per class have been queried, *Targeted* reaches 25% with labels-only, and 35% with probabilities.

Non-targeted (not shown) is behaving similarly on MNIST: training with either labels-only or probabilities yields 20% transferability with 5 seed samples per class. When 50 samples per class have been queried, *Non-targeted* reaches 48% with labels-only, and 65% with probabilities. *Non-targeted* on GTSRB already starts at 91% with labels-only, and can reach 98% with 50 seed samples per class. *Non-targeted* is already

very high, and probabilities yields at most 4 pp improvements over labels-only.

Having demonstrated the overall trend of increasing seed sample numbers and impact of probabilities, we will investigate the settings with 10 seed samples per class in more detail in the following sections.

D. Synthetic sample generation

We explore the impact that different synthetic sample crafting settings have on model extraction efficiency. As in the previous tests, we use CV-SEARCH, trained with 10 seed samples per class. At this phase, we only evaluate the scenario where the adversary has access to labels. We run model extraction attacks against MNIST and GTSRB using several synthetic sample crafting techniques (Section III-C). T-RND and COLOR techniques are used with expansion factor $k = 4$. Step size λ is set to $25.5/255$, and adversarial examples are crafted with I-FGSM.

Synthetic crafter	Test-agreement	Targeted	Non-targeted
N FGSM	0.960	0.283	0.770
N I-FGSM	-0.001	-0.006	-0.086
T-RND FGSM	**+0.008**	**+0.046**	0.008
T-RND I-FGSM	+0.007	0.043	**+0.056**
COLOR	-0.071	-0.219	-0.500

TABLE III: Impact of synthetic sample crafting strategy on model extraction performance. MNIST, 102,400 queries.

We first discuss the results for model extraction attacks on MNIST, shown in Tab. III. Non-targeted FGSM is kept as the baseline, and other methods are compared against this setup. Unsurprisingly, *Targeted* is most increased by using targeted synthetic sample crafting methods (T-RND), on average by 4.5 pp. *Non-targeted* also increases by 5.6 pp using T-RND I-FGSM. COLOR decreases *Test-agreement* and transferability over the baseline. *Test-agreement* results are already quite high for the baseline method, but targeted methods provide nearly one pp improvement over the baseline.

Synthetic crafter	Test-agreement	Targeted	Non-targeted
N FGSM	0.396	0.593	1.000
N I-FGSM	+0.056	-0.075	0.000
T-RND FGSM	-0.135	+0.002	0.000
T-RND I-FGSM	+0.112	**+0.170**	0.000
COLOR	**+0.243**	-0.498	-0.016

TABLE IV: Impact of synthetic sample crafting strategy on model extraction performance. GTSRB, 110,800 queries.

Results for the attack on GTSRB are shown in Tab. IV. *Non-targeted* is already 100% on the baseline, and none of the Jacobian-based Synthetic Sample Generation methods decrease it. We observe that creating synthetic samples using targeted methods increases *Test-agreement* and *Targeted* over baseline, with T-RND I-FGSM contributing the largest increase in *Targeted*. The largest impact of *Test-agreement* comes from the domain-specific method COLOR. We hypothesize that this is due to images occasionally having very large

| | MNIST | | | |
| | No synthetic queries | | 102,400 total queries | |
Strategy	*Test-agree.*	*Targeted*	*Test-agree.*	*Targeted*
TRAMER	-	-	6.3%	1.1%
PAPERNOT	40.0%	1.2%	95.1%	10.6%
Our T-RND-64	**82.9%**	**6.5%**	**97.9%**	**39.3%**
	GTSRB			
	No synthetic queries		110,880 total queries	
TRAMER	-	-	0.2%	2.1%
PAPERNOT	4.8%	2.4%	16.9%	41.1%
Our T-RND-64	**32.0%**	**16.9%**	47.6%	**84.8%**
Our COLOR-25	**32.0%**	**16.9%**	62.5%	27.5%

TABLE V: Comparative evaluation of model extraction attacks on our two datasets. Our techniques achieve significantly improved performance on both *Test-agreement* and *Targeted*.

differences in contrast in attacker set samples, and randomly changing colors may help in bridging the gap between attacker set images and target model training set.

We further found that large values for λ were beneficial in MNIST, both in terms of *Test-agreement* and transferability. On GTSRB, larger λ improved transferability as well, but decreased *Test-agreement*. In the next section, we evaluate these two goals separately: doing model stealing with large λ for transferability, and smaller λ for *Test-agreement*.

E. Comparative evaluation to prior work

We summarize the performance of existing model extraction techniques, and our techniques in Tab. V. We show *Test-agreement* and Transferability for our two datasets in two scenarios: using no synthetic queries and using approximately 100,000 synthetic queries. We set the number of natural seed samples to 10 per class. Transferability is evaluated with I-FGSM.

For MNIST, we find that our CV-SEARCH technique yields comparable *Targeted* as PAPERNOT even before synthetic samples are queried. With a budget of 102,400 queries, PAPERNOT reaches average *Test-agreement* 95.1%, while T-RND I-FGSM with step size $\lambda = 64/255$ reaches 97.9%, while *Targeted* and *Non-targeted* (not shown), increase to 39.9% and 87.7% compared to 10.6% and 56.2% respectively in PAPERNOT. Our techniques improve *Targeted* and *Non-targeted* on MNIST by +29.3 pp and +31.5 pp.

We see that the CV-SEARCH technique we employ is crucial for *Test-agreement* on GTSRB: PAPERNOT with 110,800 queries does not reach the same *Test-agreement* as our techniques reaches without synthetic samples. *T-RND* with step size $\lambda = 64/255$ further increases *Test-agreement* to 47.6%, while *Targeted* increases to 84.8%, compared to 41.1% in PAPERNOT. *Non-targeted* (not shown) is 100.0% in both cases. *COLOR* increases *Test-agreement* to 62.5%, while decreasing *Targeted* to 14.9%, highlighting that achieving high *Test-agreement* may be entirely complimentary to achieving transferability in model extraction attacks. TRAMER performs poorly on both datasets. This is because a large part of the random space belongs to only one class; this information is unhelpful towards building a good substitute model. Our
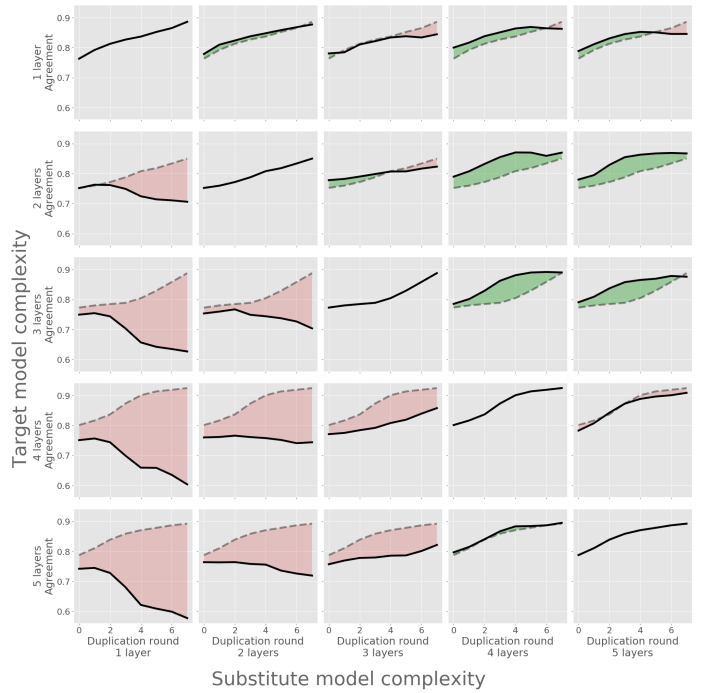

Fig. 4: Effect of architecture mismatch on *Test-agreement* (cf. Tab. VI). Columns represent increasing complexity of the substitute model (left to right). Rows represent increasing complexity of the target model (top to bottom). Substitute models with lower complexity than the target model (lower triangle, pale red) have significantly lower *Test-agreement*, compared to the baseline of matching architectures (diagonal).

techniques improve *Test-agreement* and *Targeted* on GTSRB by +46 pp and +44 pp, respectively.

F. Architecture mismatch between target and substitute models

Having seen the effect that various training strategies and synthetic sample crafting techniques have, we may ask what happens if the attacker does not know the target model architecture and instead, uses a simpler or more complex substitute model architecture than the target architecture.

We trained target models of 5 different complexities on MNIST, using architectures detailed in Tab. VI. The architectures are chosen for simplicity of evaluating increased (nonlinear) complexity. We trained each substitute model with the SAME method, and used the non-targeted FGSM for crafting synthetic samples with 7 duplication rounds. We ran the attack ten times for each of the different complexities. We show our average results for *Test-agreement* in Fig. 4. Columns denote increasing substitute model complexity as we move rightwards, and rows denote increasing target model complexity as we move downwards. The baseline is the diagonal, where model complexities match. We hilight row-wise positive change over the baseline with solid green, and negative change with pale red.

We see a clear pattern: To increase *Test-agreement*, matching or having higher model complexity than the target model

Architecture				
1 layer	2 layers	3 layers	4 layers	5 layers
				conv2-32 maxpool2
			conv2-32 maxpool2	conv2-64 maxpool2
		conv2-32 maxpool2	conv2-64 maxpool2	conv2-128 maxpool2
	FC-200	FC-200	FC-200	FC-200
FC-10	FC-10	FC-10	FC-10	FC-10
Parameters				
7,851	159,011	1,090,171	486,011	488,571

TABLE VI: Different model architectures for analysing architecture mismatch in target and substitute models. ReLU activations are used between blocks of layers. The number of parameters in the networks in reported at the bottom.

Baseline *Non-targeted* (%)				
99.4	64.6	78.4	36.3	15.5

Relative Improvement (%)				
	1	2	3	4	5
1 layer	0.0	-0.7	-34.8	-46.7	-38.4
2 layers	-70.0	0.0	-73.5	-75.5	-61.0
3 layers	-85.2	-84.9	0.0	-49.4	-52.9
4 layers	-72.2	-58.1	-24.8	0.0	-0.3
5 layers	-67.7	-67.1	-29.7	+8.4	0.0

TABLE VII: Effect of architecture mismatch on improvement on *Non-targeted* transferability. Columns represent increasing complexity of the substitute model (left to right). Rows represent increasing complexity of the target model (top to bottom). Matching architectures (diagonal) improves transferability.

is almost always beneficial for the adversary. Similarly, using a lower complexity is detrimental to the attacker, and can cause a breakdown of the attack, where *Test-agreement* drops lower than initially. This phenomena may be explained via statistical learning theory, which provides an impossibility result of perfectly reproducing a high-complexity classifier with a classifier of too low complexity, i.e. too low Vapnik-Chervonenkis (VC) dimension [34].

Non-targeted and *Targeted* transferability are also affected by model mismatch, but in a different way. We show the results for *Non-targeted* in Tab. VII, similarly evaluated on duplication round 7 using I-FGSM with step size $\epsilon = 64/255$. We see that nearly all (19 out of 20) deviations from the target model architecture cause significant decrease in the ability to produce transferable adversarial examples.

G. Takeaways

We conducted systematic, empirical tests to understand model extraction attacks on DNNs in the previous sections. We present our main observations as follows:

Hyperparameters: It is not necessary to use the same learning rate and number of training epochs as the target model was trained with. Doing CV-SEARCH can yield similar or better results for both agreement and transferability.

Seed samples: Natural seed samples are necessary to extract a substitute model that reaches high *Test-agreement*.

Synthetic sample generation: A relevant synthetic sample generation method improves transferability of adversarial examples significantly. Synthetic samples also significantly improve agreement, while remaining less efficient than using natural samples. Exploring several directions (T-RND) yields better agreement and transferability.

Training strategy: The use of probabilities rather than labels-only improves transferability for any setup, but has nearly no effect on agreement.

Mismatch between target model and substitute model: Using a higher or similar complexity substitute model as the target model architecture yields high predictive performance. Matching the architectures yields higher transferability.

Generalizability: Our dataset choices facilitated comparisons with existing methods, where they had poor *Test-agreement*: [38] on GTSRB and [55] on both. In these adversary models, adversary is not assumed to have access to *pre-trained models*: both the target and the substitute model DNNs are trained *from scratch*. Since the time of this writing, stealing DNNs for more complicated datasets like CIFAR-10[6], have been done by assuming *both* the target model and attacker models are fine-tuned from pre-trained ImageNet classifiers [37], [40]. These attacks benefit from correlations between different [40] or same [37] pre-trained models. In contrast, our paper analyzes attacks where no such correlation is present.

V. DETECTING MODEL EXTRACTION

We present PRADA (Protecting against DNN Model Stealing Attacks), a generic approach to detect model extraction attacks. Unlike prior work on adversarial machine learning defenses, e.g., for detecting adversarial examples [17], [31], our goal is not to decide whether individual queries are malicious but rather detect attacks that span several queries. Thus, we do not rely on modeling what queries (benign or otherwise) look like but rather on how successive queries relate to each other. PRADA is generic in that it makes no assumptions about the model or its training data.

A. Detection approach

We start by observing that (1) model extraction requires making several queries to the target model and (2) queried samples are specifically generated and/or selected to extract maximal information. Samples submitted by an adversary are expected to have a characteristic distribution that differs from the distribution of samples submitted in benign queries.

The distance between two randomly selected points from a totally bounded space (e.g., a cube) almost fits a normal (Gaussian) distribution [41], [48]. Inputs to a machine learning model are typically defined in a totally bounded input space, i.e., input features are defined over a certain range of values. We expect benign queries from a given client to be distributed in a natural and consistent manner. Consequently, we expect

---

[6]cf. http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/

the distance between queried samples to fit a (close to) normal distribution, as observed for random points. On the other hand, adversarial queries made to extract a model combine natural and synthetic samples coming from different distributions. Moreover, the distance between successive synthetic queries is artificially controlled by the adversary to optimally probe the input space and extract maximal information [38], [55]. Therefore, we expect the distance between adversarial queries to highly deviate from a normal distribution.

Thus, PRADA's detection method is based on detecting deviations from a normal distribution in the distance between samples queried by a given client.

(a) Benign queries (MNIST)   (b) PAPERNOT attack (MNIST)

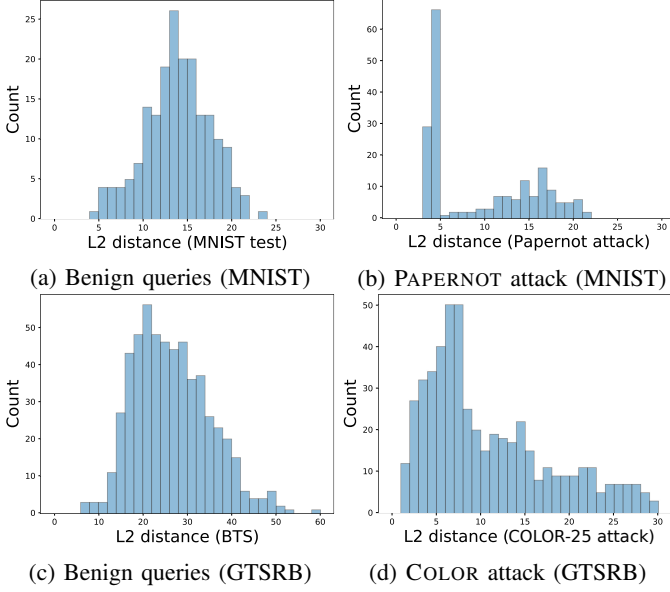(c) Benign queries (GTSRB)   (d) COLOR attack (GTSRB)

Fig. 5: Distribution of distances for benign queries and adversarial queries. **Top:** 200 queries against MNIST model, benign queries from MNIST test set (left) and PAPERNOT attack (right). **Bottom:** 600 queries against GTSRB model, benign queries from BTS dataset (left) and COLOR attack (right). Benign queries have a distribution close to normal while adversarial queries do not.

Consider the stream S of samples x queried by a single client from the target model F. We calculate the minimum distance $d_{min}(x_i)$ between a new queried sample x_i and any previous sample $x_{0,..,i-1}$ of the same class c. All $d_{min}(x_i)$ are stored in a set D. By doing so, we want to model the distribution of distances between queried samples and identify samples abnormally close to or far from any previously queried sample. For efficiency, we do not keep track of all past queries in S but incrementally build a *growing set* G_c for each class c. G_c consists only of samples whose distance d_{min} is above a threshold value T_c. We define T_c as the mean minus standard deviation of the minimum distance d_{min} between any two elements already in G_c. The distance $d_{min}(x_i)$ is computed only w.r.t. elements in G_c for $F(x_i) = c$.

Our attack detection criterion is based on quantifying how closely distances in D fit a normal (Gaussian) distribution. We

---

Algorithm 3 PRADA's detection of model extraction

1: Let F denote the target model, S a stream of samples queried by a given client, D the set of minimum distances d_{min} for samples in S, G_c the growing set for class c, D_{Gc} the set of minimum distances d_{min} for samples in G_c, T_c the threshold value for class c, δ the detection threshold.
2: $D \leftarrow \emptyset, G_c \leftarrow \emptyset, D_{Gc} \leftarrow \emptyset, attack \leftarrow false$
3: **for** $x : x \in S$ **do**
4:      $c \leftarrow F(x)$
5:      **if** $G_c == \emptyset$ **then**   # sets and threshold initialization
6:          $G_c \cup \{x\}, D_{Gc} \cup \{0\}, T_c \leftarrow 0$
7:      **else**
8:          $d \leftarrow \emptyset$
9:          **for all** $y : y \in G_c$ **do**   # pairwise distance
10:             $d \cup \{dist(y,x)\}$
11:         **end for**
12:         $d_{min} \leftarrow min(d)$ # distance to closest element
13:         $D \cup \{d_{min}\}$ # add distance to D
14:         **if** $d_{min} > T_c$ **then**   # sets and threshold update
15:             $G_c \cup \{x\}$
16:             $D_{Gc} \cup \{d_{min}\}$
17:             $T_c \leftarrow max(T_c, \overline{D_{Gc}} - std(D_{Gc}))$
18:         **end if**
19:     **end if**
20:     **if** $|D| > 100$ **then**   # analyze distribution for D
21:         $D' \leftarrow \{z \in D, z \in \langle \overline{D} \pm 3 \times std(D) \rangle\}$
22:         **if** $W(D') < \delta$ **then**   # attack detection test
23:             $attack \leftarrow True$
24:         **else**
25:             $attack \leftarrow False$
26:         **end if**
27:     **end if**
28: **end for**

---

flag an attack if the distribution of these distances deviates too much from a normal distribution. Figure 5 illustrates the intuition why our approach can effectively detect model extraction. It depicts the difference in the distribution of distances (values in D) between benign and adversarial queries. We see that benign queries to the MNIST and GTSRB models fit a distribution that is close to normal. However, adversarial queries produce spikes on several values resulting in skewed distributions. These correspond to synthetic samples for which the distance to previous samples is artificially controlled by the adversary. Other distances occur more seldom and correspond mostly to natural seed samples queried at the beginning of the attacks. Such trends are typical for all known attacks.

Several metrics exist to quantify this phenomenon and evaluate if a set of values fits a normal distribution, i.e., to perform a *normality test*. We considered and tested three, namely the Anderson-Darling test [2], the Shapiro-Wilk test [44] and the K-squared test [11]. The Shapiro-Wilk test was selected because the values of its test statistic W produced the largest difference when computed on benign and adversarial queries. A prior study also concluded that the Shapiro-Wilk test has

the most predictive power to assess whether a set of values fits a normal distribution [43]. The test statistic W used in the Shapiro-Wilk test is given in Eq. 4, where $x_{(i)}$ is the ith order statistic in the sample D, $\overline{x}$ is the sample mean, and a_i are constants related to the expected values of the order statistics. More details are provided in [44]. W is defined on $[0, 1]$ and a low value highlight deviation from a normal distribution.

$$W(D) = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2}, \text{ for } D = \{x_1, \dots, x_n\} \quad (4)$$

Algorithm 3 describes PRADA's detection technique in detail. The detection process starts when a client queries at least 100 samples ($|D| > 100$) because a sufficient number of values is necessary to compute a relevant W. We first remove outliers from D, i.e., values being more than 3 standard deviations away from the mean of values in D. According to the *68-95-99.7* empirical rule, 99.7% of values coming from a normal distribution belong to this interval. We compute the Shapiro-Wilk test statistic W on the resulting set deprived from outliers D'. Next, if $W(D')$ is below a threshold δ, PRADA detects an extraction attack.

PRADA requires the defender to set one parameter: the detection threshold δ. It also needs a domain-specific distance metric $dist()$ to compute distances between inputs. We use L^2 (Euclidean) norm for image samples in our experiments.

B. Evaluation

We evaluate PRADA in terms of success and speed. Speed refers to the number of samples queried by an adversary until we detect the attack. It correlates with the amount of information extracted and must be minimized. We also evaluate the *false positive rate* (FPR): the ratio of false alarms raised by our detection method to all query sequences from benign clients.

To evaluate success, we assess its detection of attacks against the two target models previously trained in Sect. IV-A for MNIST and GTSRB datasets. We subject these models to four different attacks: TRAMER, PAPERNOT and our new *IFGSM* T-RND-64 (noted T-RND here) and COLOR-25 attack (noted COLOR here). We use the samples generated while evaluating the performance of these attacks in Sect. IV and query the prediction model with them one by one (in the order they were generated). PRADA's detection algorithm is computed for each new queried sample. When an attack is detected, we record the number of samples queried until then by the adversary to evaluate the speed of detection.

To evaluate the false positive rate, we use natural data from MNIST and GTSRB datasets. To demonstrate that PRADA is independent of a specific data distribution, we also use randomly generated samples (images with uniformly random pixel values), the U.S. Postal Service (USPS) [27] and Belgian traffic signs (BTS) [52] datasets. USPS and BTS datasets contain similar data as MNIST and GTSRB respectively but from different distributions. We reshaped the samples to fit the input size of MNIST and GTSRB models. We simulate

a benign client by randomly picking 6,000 samples from a given dataset and successively querying the appropriate model: *MNIST/USPS/random* → MNIST model, *GTSRB/BTS/random* → GTSRB model. We simulate five benign clients per dataset (30 clients). To evaluate FPR, we split this sequence of queries into 120 chunks of 50 queries each and count a false positive if at least one alert is triggered by PRADA in a chunk. Successive benign queries can also be related to each other. In a self-driving car scenario, successive pictures of the same road sign are taken and submitted to the model while getting closer to it. We simulated this scenario using the GTSRB validation set that contains 207 *sequences* composed of 30 pictures each of a single road sign taken from a decreasing distance (6,210 samples). We ran five tests, randomly shuffling the order of sequences and submitting them to the GTSRB model while computing the FPR.
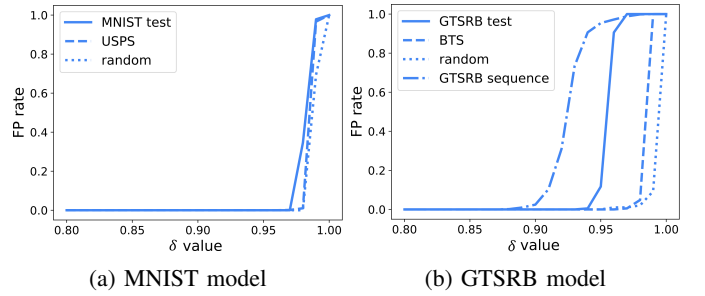

Fig. 6: FPR for PRADA vs. detection threshold δ.

Figure 6 depicts the increase in FPR according to the detection threshold value δ. A high δ value of 0.96 results in no false positives for MNIST model while GTSRB model requires $\delta = 0.87$ to reach the same result. It is worth noting that different simulated benign users generate queries with different distributions in the GTSRB experiment (Fig. 6b). While the *BTS* and *random* queries are distributed close to normal (high δ values result in no false positives), GTSRB queries are further away from a normal distribution (lower δ values required for no false positives). In the *GTSRB sequence* queries, images in a sequence have a relatively small distance to each other while images from different sequences have higher distances between them. This explains a more scattered distribution of distances that deviates from a normal distribution. This shows that δ is a domain specific parameter that needs to be set with respect to the model to protect and its use case scenario.

Table VIII presents detailed speed of detection for a few selected δ values. Most attacks are detected shortly after they have a change in their query strategy. From natural to synthetic samples for PAPERNOT, T-RND and COLOR attack (after 100 queries for MNIST and 430 queries for GTSRB). From random samples to line search strategy for TRAMER attack (after 5,000 queries). While the detection is slower for the TRAMER this is not a concern since it is itself slow in extracting DNNs. An estimate for the performance of the substitute model at the time of detection can be found in Tab. V (no synthetic queries). The T-RND attack is the only one that

TABLE VIII: Adversarial queries made until detection respect to δ value. FPR is averaged over all simulated benign query scenarios (* = *GTSRB sequence* was discarded). COLOR attack is only performed against the GTSRB model so no results are reported for MNIST.

Model (δ value)	FPR	Queries made until detection			
		TRAMER	PAP.	T-RND	COLOR
MNIST (0.95)	0.0%	5,560	120	140	-
MNIST (0.96)	0.0%	5,560	120	130	-
GTSRB (0.87)	0.0%	5,020	430	missed	550
GTSRB (0.90)	0.6%	5,020	430	missed	480
GTSRB (0.94)	0.1%*	5,020	430	440	440

remains undetected against the GSTRB model if δ is too low. This is because the it uses a large step size $\lambda = 64$ which produces synthetic samples with a large distance between each other. This distance happens to fit the normal distribution of natural samples for the GTSRB model. By increasing δ to 0.94, PRADA effectively detects the T-RND attack while producing a few false positives (0.1%). This δ value cannot be applied to all scenarios though, e.g., it triggers a large number of false positives with *GTSRB sequence* queries (cf. Fig. 6b).

For most of our tests, this demonstrates that PRADA is effective at protecting against most model extraction attacks developed to date. Using an appropriate δ threshold, it detects quickly TRAMER, PAPERNOT and COLOR attacks while avoiding false positives for benign queries across the tested datasets: MNIST, USPS, GTSRB (+ sequence), BTS and random queries. A more careful selection of δ is necessary to detect the T-RND attack against GTSRB, and it may not apply to any model deployment scenario (e.g., high FPR in *sequence* scenario), meaning that PRADA perhaps cannot be reliably deployed in all scenarios, as it may limit the usability.

To estimate the overhead of PRADA, we computed the memory required to store the growing set G. Note that G represents a subset of all queries S. Samples for MNIST and GTSRB models have an average size of 561B and 9.3kB respectively. The average memory required before detecting PAPERNOT and *T-RND* attack for MNIST is around 55kB ($561B \times 98$ samples) and 3.2MB for GTSRB ($9.3kB \times 343$ samples). Benign clients generate a larger G since its growth is not stopped by a detection. However, this growth naturally slows down as a client makes more queries. As an estimate, we used 1.9MB (MNIST test: $561B \times 3,374$ samples) and 1.0MB (USPS: $561B \times 1,804$ samples) for storing G of a MNIST model client submitting 6,000 queries. We used 28.1MB (GTSRB test: $9.3kB \times 3,025$ samples) and 30.2MB (BTS: $9.3kB \times 3,254$ samples) for storing G of a GTSRB model client submitting 6,000 queries.

C. Discussion

Evasion of Detection: We observed that PRADA can be evaded by PAPERNOT and T-RND type of attacks by carefully selecting a step size λ that would simulate a normal distribution of samples (cf. Sect. V-B T-RND attack against GTSRB). We also concluded that λ is a important factor impacting the
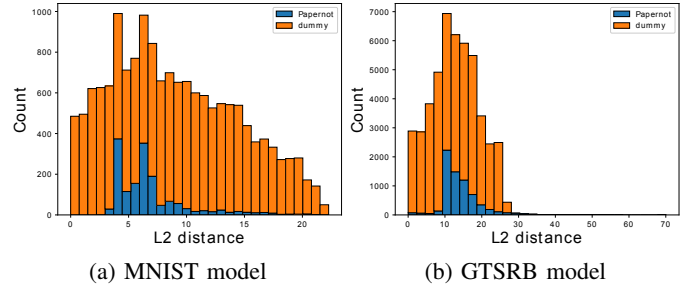

(a) MNIST model   (b) GTSRB model

Fig. 7: Example of controlling the distribution of D under PAPERNOT attack. **Left:** 1,600 attack queries (blue) against MNIST model. 14,274 dummy queries (orange) are required to avoid detection. **Right:** 6,680 attack queries against GTSRB model, 41,160 dummy queries are required to avoid detection.

TABLE IX: Increased query cost to circumvent PRADA (**GTSRB** T-RND * with $\delta = 0.94$).

Model	MNIST ($\delta = 0.96$)		
Attack	PAPERNOT	*T-RND*	TRAMER
Original queries	1,600	1,600	10,000
Additional queries	14,274	4,764	79,980
Overhead	+890%	+300%	+800%

Model	GTSRB ($\delta = 0.90$)			
Attack	PAPERNOT	T-RND *	TRAMER	COLOR
Original queries	6,680	6,680	10,000	6,680
Additional queries	41,160	19,986	99,990	39,972
Overhead	+620%	+300%	+1000%	+600%

success of a model stealing attack (cf. Sect IV-D). Evading PRADA by modifying the step size λ optimal for model stealing purposes may degrade the performance of the stolen model.

Alternatively, an adversary can attempt to evade detection by making *dummy queries* that are not useful for building the substitute model but that would maintain a normal distribution of distances between queries. To evaluate the additional query cost of this evasion attack, we simulated an adaptive adversary. Following Kerckhoffs's Principle, it has full knowledge of the detection algorithm, including the detection threshold value δ that is supposed to be secret.

In order to control the query distribution, a dummy query needs to satisfy two conditions: (1) it must be compared to a selected subset G_c of the *growing set* (targeted classification) and (2) its minimum distance d_{min} (cf. Alg. 3) must help to avoid detection, i.e., increase $W(D)$ to satisfy $W(D) \geq \delta$. Dummy queries must complement D to fit a normal distribution as shown in Fig. 7. Table IX presents the query overhead required to circumvent PRADA. It ranges from $+3\times$ to $+10\times$ more queries depending on the target model and attack considered.

In our evaluation, we controlled the distribution of D using selected d_{min} values and without generating queries. Thus our evaluation provides an estimated lower bound on the number of queries required to circumvent PRADA. We experimented

with several strategies concluding that creation of such queries is not trivial. Notably, the following strategies did not fool our detection algorithm:

1) random noise drawn from normal/uniform distribution
2) natural samples of desired class perturbed with random noise
3) like *(1)* and *(2)* but constraining d_{min} to be in range $\langle \overline{D} \pm std(D) \rangle$ or $\langle \overline{D} \pm 2 \times std(D) \rangle$
4) like *(1)* and *(2)* but submitting the sample only if it satisfies the $W(D) \geq \delta$.

In case of strategies *(1)* and *(2)* we observed that *dummy queries* and *useful queries* formed two spaced out peaks in the distribution. For *(3)* the underlying seed samples impacted d_{min} too much unless the noise was large enough to be equivalent to *(1)* or *(2)*. Finally, in *(4)*, after several samples (≈ 50) the search for individual images became too time consuming (thousands of samples) to find a single valid query.

Since PRADA analyses samples queried by a single client, an adversary can distribute its queries among several clients to avoid detection (*Sybil attack*). Using a sufficient number of clients, PRADA can be circumvented.

Countermeasures: Once PRADA detects an attack, we must resort to effective mitigation. Blocking requests from the adversary would be a straightforward prevention. This would be effective on single-client models protected by local isolation. The defender might also deceive the adversary with altered predictions once an attack is detected in order to degrade the substitute model learned by the adversary. Returning the second or third class with the highest likelihood according to the prediction of the target model may plausibly deceive the adversary into thinking it has crossed a class boundary while it has not and effectively undermine its substitute model.

Generalizability: PRADA is applicable to many types of data and ML models without any alterations since its design is independent from these considerations and only relies on identifying adversarial querying behavior. The only aspect that depends on the type of data is finding a distance metric appropriate to compute differences between input samples of a certain type, e.g., we chose L^2 norm for image input. Alternatively, any L^p norm or the structural similarity metric [45] could be used for image input. As examples for other domains, the decibel metric (*dB*) can be used on audio input [7] and the L^1 norm on malware input [18].

One must also set an appropriate detection threshold δ. This is dependent on the use case scenario for the model which will define a "benign" distribution of queries, as highlighted in Sect. V-B. This value can be fixed using a training period during which only benign queries are submitted to the system and δ is selected as a maximum value that does not generate any false positives as we showed in Fig. 6. Capturing this benign distribution correctly will impact the detection efficacy of PRADA.

Storage overhead and scalability: PRADA requires keeping track of several client queries, substantially increasing memory consumption. It is worth noting that we presented results for the extreme case of image classification models, which use high dimensional inputs. Nevertheless the amount of memory required per client was estimated to be a few megabytes (1-30 MB), which is reasonable. For local models being used by single clients, the storage requirements are thus minor. Multi-client remote models serving up to a few hundred clients simultaneously will require a few gigabytes of memory in total. This is reasonable for a cloud-based setup where the model is hosted on a powerful server.

VI. RELATED WORK

A. Model Extraction Attacks

Model extraction is conceptually similar to concept learning [3], [5] in which the goal is to learn a model for a concept using membership queries. Concept learning has been used for adversarial machine learning and evading binary classifiers [30], [35]. Model evasion may be considered a theoretically simpler task than model extraction [51], and so far, the efficiency of model extraction attacks have not been demonstrated on DNNs. The extraction of information from DNNs has been addressed in non-adversarial settings by compressing DNNs to simpler representations [6], [19] or by obtaining interpretable decisions from ML models [10], [53]. These work do not apply to adversarial settings since they require white-box access to the target model and its training data.

We presented the two closest prior works to ours in Sect. II-F. Tramer et al. [55] introduced several methods for extracting ML models exposed in online prediction APIs. They exploit the confidence values from predictions in a systematic equation solving approach to infer exact model parameters. In contrast to our attacks, this method applies only to simple models and it is ineffective at extracting DNN models (cf. Sect. IV-D). Papernot et al. [38] introduced a method for extracting a substitute DNN model for the specific purpose of computing transferable non-targeted adversarial examples. Their main contribution is the JbDA technique for generating synthetic samples (cf. Sect. II-F).

In contrast to these works, we introduce a generic method for extracting DNNs. It is multipurpose and has higher performance in transfer of targeted adversarial examples and reproduction of predictive behavior.

Alternative model stealing attacks assume access to large sets of natural samples and use active learning strategies to select the best samples to query [37]. In this paper, we consider a different adversary model with limited access to natural samples. A recent line of work targets the extraction of model hyperparameters and architecture. Joon et al. [36] train a supervised classifier taking as input n prediction values rendered by a classifier for a fixed set of n reference samples. Using this technique, they infer with significant confidence the architecture, optimization method, training data split and size, etc. of a confidential target model. Hua et al. [20] target a model locally isolated with hardware security mechanisms (Intel SGX) and introduce a hardware side channel attack to infer similar model information. Another work [56] takes a stronger adversary model (access to training data) and introduces a

technique for computing the value for the hyperparameter for L^2-regularization. [36] is complementary to our attack and can be used in the first stage to select the architecture for the substitute model.

B. Defenses against Model Extraction

A first defense to model extraction is to reduce the amount of information given to an adversary by modifying the model prediction. Prediction probabilities can be quantized [55] or perturbed to deceive to the adversary [29]. We have shown that model extraction attacks are effective even without using prediction probabilities (Sect. IV), making this line of defenses ineffective. A second line of defense consists in detecting model extraction attacks by recording requests from clients and computing the feature space explored by the aggregated requests [23]. When the explored space exceeds a predetermined threshold, an extraction attack is detected. Quiring et al. [42] use the same intuition and study the closeness of queries to class boundaries to detect model stealing attacks. These techniques have limitations since they require linearly separated prediction classes (both are applied to decision trees). Thus they do not apply to high dimensional input spaces nor to DNN models, which build highly non-linear decision boundary in this space. The false alarm rate of this technique is not evaluated and might be high since a legitimate client can genuinely explore large areas of the input space. On the contrary, PRADA applies to any input data dimensionality and any ML model. It is effective at detecting model extraction attacks developed to date and does not degrade the prediction service provided to benign clients.

Alternatively, methods for detecting adversarial examples can help detecting synthetically generated samples from *Papernot* attack and ours. The main approaches rely on retraining the model with adversarial samples [54], randomizing the decision process [14] or analyzing the inputs distribution [31]. These techniques assume a specific distribution of the benign inputs to the prediction model, i.e., the same distribution as the training data. Consequently, they may raise a high number of false alarms if benign clients request natural samples distributed differently than the training data.

In contrast, PRADA has been developed in mind of avoiding false positives. It does not assume any training data distribution but only studies the evolution in distribution of samples submitted by a given client. This explains why we have low or no false positives even when analyzing benign data from diverse distributions. Methods for detecting adversarial examples may not generalize to detected the *Tramer* class of attacks [55] since it does not rely on methods for crafting adversarial examples.

VII. CONCLUSION

We have systematically explored approaches for model extraction. We evaluated several attacks on different DNN models and showed that hiding hyperparameters of the target model does not help protect against model extraction. Reducing DNN outputs from classification probabilities to labels only has nearly no impact on prediction accuracy, but does impact transferability of adversarial examples. Keeping model architectures confidential helps to protect against model extraction attack and transferable adversarial examples. In scenarios where it is possible, limiting the adversary's access to natural seed samples, can also limit the effectiveness of model extraction.

Recent research has shown that ML models, especially DNNs, suffer from various vulnerabilities. Consequently, protecting confidentiality of models is a useful mitigation. In this black-box scenario, an attacker is forced to repeated interactions with the model. We demonstrated that model extraction can be effectively detected by collecting *stateful* information of queries in *ML prediction APIs*. This defense has significant advantages since it does not require any knowledge about the ML model, nor about the data used to train it. Relying on deviations from benign distributions, we found it can be circumvented if the attacker mimics such distributions. We leave robustness against such an attacker to future work. Model confidentiality combined with a stateful defense strategy is a promising venue for effectively protecting ML models against a large range of adversarial machine learning attacks. One example we are currently exploring is defending against black-box attacks for forging adversarial examples (without resorting to building substitute models via model stealing attacks; see Appendix). Such attacks usually require thousands of queries to forge one adversarial example. A stateful prediction API like the one described in this paper with PRADA appears to be a promising defense direction.

REFERENCES

[1] Amazon, "Amazon machine learning," https://aws.amazon.com/aml/, last accessed 14/11/2018.

[2] T. W. Anderson and D. A. Darling, "Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes," *The annals of mathematical statistics*, 1952.

[3] D. Angluin, "Queries and concept learning," *Machine learning*, vol. 2, no. 4, 1988.

[4] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[5] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, "Oracles and queries that are sufficient for exact learning," *Journal of Computer and System Sciences*, vol. 52, no. 3, 1996.

[6] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of ACM SIGKDD*, 2006.

[7] N. Carlini and D. A. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," *CoRR*, vol. abs/1801.01944, 2018.

[8] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of ACM AISec*, 2017.

[9] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine learning*, vol. 15, no. 2, 1994.

[10] M. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," in *Proceedings of NIPS*, 1996.

[11] R. B. d'Agostino, "An omnibus test of normality for moderate and large size samples," *Biometrika*, vol. 58, no. 2, 1971.

[12] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Computer Vision and Pattern Recognition*, 2018.

[13] J. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security & Privacy*, vol. 12, no. 4, 2014.

[14] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *CoRR*, vol. abs/1703.00410, 2017.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016, vol. 1.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[17] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.

[18] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *ESORICS*. Springer, 2017.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[20] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *ACM Design Automation Conference*, 2018.

[21] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Query-efficient black-box adversarial examples," *arXiv preprint arXiv:1712.07113*, 2017.

[22] Intel, "Movidius myriad x vpu," https://www.movidius.com/, last accessed 14/11/2018.

[23] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta, "Model extraction warning in MLaaS paradigm," in *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018.

[24] J. Konecný, B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of NIPS*, 2012.

[26] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[27] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proceedings of NIPS*, 1990.

[28] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs*, 2010.

[29] T. Lee, B. Edwards, I. Molloy, and D. Su, "Defending against model stealing attacks using deceptive perturbations," *arXiv preprint arXiv:1806.00054*, 2018.

[30] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of ACM SIGKDD*, 2005.

[31] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proceedings of ACM CCS*, 2017.

[32] Microsoft, "Azure machine learning," https://azure.microsoft.com/en-us/overview/machine-learning/, last accessed 14/11/2018.

[33] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of ACM AISec*, 2017.

[34] K. Murphy, "Machine learning: a probabilistic approach," *Massachusetts Institute of Technology*, 2012.

[35] B. Nelson, B. I. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. Tygar, "Query strategies for evading convex-inducing classifiers," *Journal of Machine Learning Research*, vol. 13, no. May, 2012.

[36] S. J. Oh, M. Augustin, M. Fritz, and B. Schiele, "Towards reverse-engineering black-box neural networks," in *International Conference on Learning Representations*, 2018.

[37] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[38] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of ACM ASIACCS*, 2017.

[39] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[40] L. Pengcheng, J. Yi, and L. Zhang, "Query-efficient black-box attack by active learning," in *IEEE 2018 International Conference on Data Mining*. IEEE, 2018.

[41] J. Philip, *The probability distribution of the distance between two random points in a box*. KTH mathematics, Royal Institute of Technology, 2007.

[42] E. Quiring, D. Arp, and K. Rieck, "Forgotten siblings: Unifying attacks on machine learning and digital watermarking," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 488–502.

[43] N. M. Razali, Y. B. Wah *et al.*, "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests," *Journal of statistical modeling and analytics*, vol. 2, no. 1, 2011.

[44] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, 1965.

[45] M. Sharif, L. Bauer, and M. K. Reiter, "On the suitability of lp-norms for creating and preventing adversarial examples," in *Proceedings of IEEE CVPR Workshops*, 2018.

[46] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proceedings of NIPS*, 2017.

[47] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of NIPS*, 2012.

[48] L. A. S. Sors and L. A. Santaló, *Integral geometry and geometric probability*. Cambridge university press, 2004.

[49] N. Šrndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *IEEE Symposium on Security and Privacy*, 2014.

[50] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: a multi-class classification competition," in *IEEE International Joint Conference on Neural Networks*, 2011.

[51] D. Stevens and D. Lowd, "On the hardness of evading combinations of linear classifiers," in *Proceedings of ACM AISec*, 2013.

[52] R. Timofte, K. Zimmermann, and L. van Gool, "Multi-view traffic sign detection, recognition, and 3d localisation," in *IEEE Computer Society Workshop on Application of Computer Vision*, 2009.

[53] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine learning*, vol. 13, 1993.

[54] F. Tramer, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[55] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis." in *USENIX Security Symposium*, 2016.

[56] B. Wang and N. Zhenqiang Gong, "Stealing Hyperparameters in Machine Learning," in *IEEE Symposium on Security and Privacy*, 2018.

APPENDIX

We defined *black-box* adversaries with *surrogate data* in our paper. In addition, the following *black-box* adversarial attacks have been examined in literature:

a) Surrogate Learner: This setting is similar to ours, in that a substitute model is used for the adversarial attacks. However, Munoz et al. [33] state that this threat model does not assume knowledge on what type of target classifier is used, but may use *same training data*. Adversarial examples for ImageNet models [25] are typically shown in this setting, e.g. Dong et al. [12] show it is possible to create highly transferable adversarial examples.

b) Finite difference methods: It is also possible to create targeted adversarial examples for DNNs without substitute models [8], [21]. These attacks are very effective, but have limitations: these attacks require *thousands of queries per sample* and may be therefore easily detectable, they do not extract models and mostly require access to target model probabilities. We calculated that attacking MNIST with Natural Evolution Strategies [21] requires on average several 1000s queries on MNIST, and several 100s queries on GTSRB per adversarial example.