HODA: Hardness-Oriented Detection of Model Extraction Attacks

Amir Mahdi Sadeghzadeh, Amir Mohammad Sobhanian, Faezeh Dehghan, and Rasool Jalili

Abstract—Model extraction attacks exploit the target model’s prediction API to create a surrogate model, allowing the adversary to steal or recompute the functionality of the target model in the black-box setting. Several recent studies have shown that a data-limited adversary with no or limited access to the samples from the target model’s training data distribution, can employ synthesized or semantically similar samples to conduct model extraction attacks. In this paper, we introduce the concept of hardness degree to characterize sample difficulty based on the concept of learning speed. The hardness degree of a sample depends on the epoch number at which the predicted label for that sample converges. We investigate the hardness degree of samples and demonstrate that the hardness degree histogram of a data-limited adversary’s sample sequence is different from that of benign users’ sample sequences. We propose Hardness-Oriented Detection Approach (HODA) to detect the sample sequences of model extraction attacks. Our results indicate that HODA can effectively detect model extraction attack sequences with a high success rate, using only 100 monitored samples. It outperforms all previously proposed methods for model extraction detection.

Index Terms—Model extraction, model stealing, adversarial machine learning, hardness of samples, adversarial examples.

I. INTRODUCTION

Deep Neural Networks (DNNs) have shown impressive performance in various tasks in recent years that have encouraged the industry to deploy DNN-based models in a variety of real-world applications. Since the training process of DNNs and collecting training data is an expensive and tedious process, models are considered the intellectual property of organizations, and they must be kept secure. Therefore, models are often securely deployed on cloud servers, and only the creators can access the model parameters. Users are only allowed to query the model via a prediction API and receive predictions. Recent studies [1], [2], [3], [4], [5] demonstrate that an adversary can exploit the prediction API of a target model to create a surrogate model in order to steal or recompute the functionality of the target model. Such attacks are called model extraction attacks, and they violate the intellectual property of model owners. Furthermore, the surrogate model can be leveraged to conduct other attacks on the target model in the black-box setting, such as adversarial example attack [2], [3], membership inference attack [6], and model inversion attack [7].

Most of model extraction attacks use the target model’s prediction API to label an unlabeled dataset to create the surrogate model’s training set. In most real-world settings, the adversary has no or limited access to samples from the target model’s training data distribution, which is called normal or in-distribution samples. Hence, most proposed attacks in the previous studies use some form of Out-Of-Distribution (OOD) samples, such as synthesis [2], [3] or semantically similar samples to the target model’s training set [4], [8] to conduct model extraction attacks. We focus on such attacks in this paper. There are two main approaches to defend against model extraction attacks, manipulating the target model outputs to prevent adversary from producing high-quality surrogate model [9], [10], [11], [12] and detecting the sample sequences of model extraction attacks [3], [13], [14], [15], [16]. We propose Hardness-Oriented Detection Approach (HODA) in order to detect sample sequences of model extraction attacks. HODA outperforms PRADA [3], VarDetect [14], and TMSPDetect [16] by a large margin.

In this paper, we use the concept of learning difficulty [17], [18] to define our hardness measure. We expect the predicted label of easy samples to converge sooner than the predicted label of hard samples during the course of training. Hence, the hardness degree of a sample depends on the predicted label convergence speed for that sample during training. We consider a DNN-based classifier training process as a sequence of sub-classifiers so that each sub-classifier is created at the end of an epoch. HODA uses a subsequence of sub-classifiers to compute the hardness degree of samples. Since attack samples of a data-limited adversary do not lie on the data manifold that is well supported by the target classifier during training, attack samples have a very small number of easy samples, unlike normal samples. Normal samples are independent and identically distributed (i.i.d.) data from the target classifier’s training data distribution. We demonstrate that the hardness degree histogram of benign user’s samples is distinguishable from the hardness degree histogram of model extraction attack samples. HODA uses this observation to detect sample sequences of model extraction attacks. For each user, HODA computes the distance between the hardness degree histograms of the user’s samples and normal samples, and if the distance exceeds a threshold, the user is detected as an adversary. HODA can detect JBDA [2], JBRAND [3], and Knockoff Net [4] attacks with a high success rate by only...
monitoring 100 samples of attack. We demonstrate that HODA is also highly effective on high-dimensional datasets when the target classifier is trained using transfer learning.

The main contributions of this paper are as follows:

- We demonstrate that the hardness degree of a sample for a classifier pertains to the training data distribution of that classifier.
- We demonstrate that the misclassification rate increases as the hardness degree of samples increases.
- We indicate that there is a small number of easy samples among model extraction attack samples.
- We propose HODA to detect sample sequences of model extraction attacks. It outperforms all previous model extraction detection methods and is more scalable than them. Also, HODA is effective on high-dimensional datasets.

II. PRELIMINARIES

A. Deep Neural Networks

A Deep Neural Network (DNN)-based classifier is a hierarchical function \( f : \mathcal{X} \rightarrow \mathcal{Y} \) that maps the input space \( \mathcal{X} \) to the output space \( \mathcal{Y} \). Each DNN has a parameter set \( \theta \) being tuned during the training process to minimize the loss function \( \mathcal{L} \) on the training set \( \mathcal{X} \). The training set \( \mathcal{X} = \{(x_i, y_i)\}_{i=1}^{N} \) consists of \( N \) pairs that each pair includes a sample \( x_i \in \mathbb{R}^d \) and the corresponding label \( y_i \in [K] \), where \( d \) is the input dimension, and \( K \) is the number of classes. The output of classifier \( \hat{y}_i = f(x_i) \) is a probability vector over \( K \) classes. The \( j^{th} \) element of \( \hat{y}_i \) indicates the classifier’s confidence that \( x_i \) belongs to the \( j^{th} \) class. Hence, the label of data \( x_i \) is \( f(x_i) = \text{argmax}(f(x_i)) \). The loss function \( \mathcal{L} \) is used to calculate the distance between label \( y_i \) and the output of classifier \( \hat{y}_i = f(x_i) \) for sample \( x_i \). The minimization of loss function \( \mathcal{L} \) is usually done using some common versions of the Stochastic Gradient Descent (SGD). SGD is an iterative algorithm that initializes the classifier’s parameters randomly and then takes a step in the inverse direction of the gradient of loss function with respect to the classifier’s parameters in each iteration. Most often, SGD is run for several epochs. All samples in the training set are shuffled and then, are partitioned into several mini-batches in each epoch. SGD uses each mini-batch to calculate the loss function and updates classifier’s parameters for one step.

B. Model Extraction Attacks

The model extraction attack is one of the most serious threats against machine learning-based classifiers on remote servers, such as Machine Learning as a Service (MLaaS). The adversary’s goal is to create a surrogate classifier \( f_s \) that imitates target classifier \( f_t \) on task \( T \). Most model extraction attacks exploit target model \( f_t \) to label a set of unlabeled samples to create the surrogate model’s training set. The adversary sends sample \( x_i \) to the target model and receives its output \( f_t(x_i) \), and then uses pair \( (x_i, f_t(x_i)) \) to train surrogate classifier \( f_s \). Proposed model Extraction attacks create the surrogate classifier training set \( \mathcal{X}_s = \{(x_i, f_s(x_i))\}_{i=1}^{B} \) by various methods, where \( B \) is the attack budget. The attack budget determines the number of samples that an adversary is allowed to send to the target classifier and receive their associated predictions. After creating \( \mathcal{X}_s \), the adversary trains surrogate classifier \( f_s \) to minimize empirical loss on \( \mathcal{X}_s \). We suppose that the adversary knows the architecture and hyperparameters of the target classifier and uses them to train the surrogate classifier. It is noteworthy that our proposed defense is independent of surrogate classifier training process. The output type of target model can be label, label confidence, top-k values in probability vector, or the entire probability vector. We only consider label \( f_t(x_i) \) and the entire probability vector \( f_t(x_i) \) as the output type of target classifiers in our experiments. There are two primary intents for adversaries to conduct model extraction attacks, stealing and reconnaissance [5]

1) Stealing: Producing a DNN-based classifier is an expensive and time-consuming process and requires labeled dataset, computational resources, and experts. Therefore, adversaries are motivated to take advantage of a target classifier to reduce the cost of creating a new classifier. The adversary’s goal in stealing is to maximize the accuracy of surrogate model on data distribution \( \mathcal{D}_T \). Hence, the adversary’s goal is:

\[
\text{Maximize} \quad P_{(x,y) \sim \mathcal{D}_T} \bar{f}_s(x) = y \quad (1)
\]

2) Reconnaissance: The model extraction attacks can be used to conduct other attacks in the black-box setting, such as adversarial example attack [2], [3], membership inference attack [6], and model inversion attack [7]. The adversary’s goal in reconnaissance is to maximize the fidelity among surrogate and target classifiers in order to increase the success rate of black-box attacks. Similar to [5], we consider label agreement among surrogate and target classifiers as the fidelity metric on data distribution \( \mathcal{D}_T \). Hence, the adversary’s goal is:

\[
\text{Maximize} \quad P_{(x,y) \sim \mathcal{D}_T} \bar{f}_s(x) = \bar{f}_r(x) \quad (2)
\]

Most model extraction attacks in the literature use some form of out-of-distribution samples to conduct model extraction attacks for two primary reasons. First, adversaries are motivated to conduct model extraction attacks with out-of-distribution samples to decrease the cost of procuring, cleaning, and pre-processing in-distribution samples [4], [8], [19], [20], [21], [22], [23]. As the cost and the difficulty of collecting in-distribution samples increase, adversaries are more motivated to conduct model extraction attacks. Second, adversaries use out-of-distribution samples to increase the fidelity of the surrogate classifier to the target classifier in order to conduct more advanced attacks on the target model in the black-box setting [1], [2], [3], [5].

III. RELATED WORK

A. Model Extraction Attacks

Primary model extraction attacks try to extract the exact value of parameters [1], [24] and hyperparameters [25] of shallow models. For the first time, Papernot et al. [2] propose Jacobian-Based Dataset Augmentation (JBDA) attack for a deep neural network in order to generate adversarial examples in the black-box setting. The goal of JBDA attack is to increase the fidelity of the surrogate classifier to the target
classifier in order to produce adversarial examples for the target classifier in the black-box setting using transferability property of adversarial examples. The authors assume that the adversary has access to a limited number of normal samples called seed samples. JBDA augment seed samples using adversarial examples to improve the fidelity of surrogate classifier to the target classifier. The augmentation process is conducted in multiple rounds. In the first round, surrogate training set \( X_s \) is initialized by seed samples, and surrogate model \( f_s \) is trained on \( X_s \). In the next rounds, sample set \( S \) with size \( \kappa \) is randomly selected from \( X_s \), and for each \( x \in S \), adversarial example \( x' \) is created using the following equation:

\[
x' = x + \lambda \cdot \text{sign}(J_f(x))
\]

where \( \lambda \) is step size and \( J \) is the Jacobian function. Afterward, new adversarial examples are labeled by the target model, and they are added to \( X_s \). Lastly, surrogate model \( f_s \) is trained on \( X_s \). Juuti et al. [3] propose JBRAND attack to improve the performance of JBDA. JBRAND perturbs each sample in multiple iterations to generate more powerful adversarial examples and generates targeted adversarial examples with random targets. Knockoff Net [4], ActiveThief [8], and Copycat CNN [19] suppose that adversaries have no access to normal samples and they use large public datasets that have semantically similar samples to the target classifier training set in order to create surrogate classifier training set. For example, they send a subset of ImageNet dataset [26] to the target image classifier and receive the corresponding predictions, and then, train a surrogate classifier on the samples in the subset and the corresponding target model predictions. They employ different strategies for selecting \( B \) (budget) samples from attack datasets to extract more information from the target classifier. Knockoff Net attack [4] has adaptive and random strategies to select the surrogate classifier training set from attack datasets. Adaptive strategy uses reinforcement learning to select attack samples. In random strategy, an adversary randomly selects a subset of a public dataset and labels them using the target classifier to create surrogate classifier training set. The accuracy of surrogate classifier in adaptive strategy is slightly better than random strategy. However, adaptive strategy has high computational cost. Yu et al. [20] employ active learning, transfer learning, and a new method for generating adversarial examples to improve model extraction attacks efficiency. A line of studies [21], [22], [23] use synthetic data to create the training set of surrogate classifiers. Although their methods do not need to have access to natural samples, they send a high number of queries to the target classifier, which makes their methods impractical. For example, [21] and [22] send millions of queries to extract a target classifier trained on CIFAR10 dataset. Jagielski et al. [5] use semi-supervised learning methods to improve the performance of model extraction attacks.

B. Defenses Against Model Extraction Attacks

Existing defense methods against model extraction attacks are generally distributed into two branches: perturbation-based and detection-based defenses. Perturbation-based defenses [9], [10] attempt to prevent adversaries from producing high-quality surrogate classifiers by adding perturbation to the output of target classifier. These methods generate perturbations with various strategies to minimize the accuracy of surrogate classifiers. If an adversary only uses the predicted label of attack samples rather entire probability vector, these defense methods must decrease the accuracy of the target classifier to be effective. To preserve the accuracy of target classifier on in-distribution samples, Kariyappa and Qureshi [11] and Kariyappa et al. [12] propose defenses that change the output of target classifier only for Out-Of-Distribution (OOD) samples. To detect OOD samples, they use classifiers that have been trained on OOD samples that are similar to samples that a data-limited adversary use to conduct model extraction attacks. Hsu et al. [27] demonstrate that using OOD dataset to train an OOD detector can easily bias the learning. Perturbation-based defenses can not prevent adversaries from conducting model extraction attacks, and they only decrease the quality of the surrogate classifier.

Detection-based defenses [3], [13] attempt to detect the occurrence of model extraction attacks by observing successive input queries to the target classifier. PRADA [3] is the first proposed detection-based defense for DNN models. PRADA uses the histogram of the minimum \( L_2 \) distance among a new sample and all previous samples to detect model extraction attacks. Aside from its high computational overhead, it has been shown that PRADA is unable to detect model extraction attacks that use natural samples [8], such as Knockoff Net. Atlı et al. [28] demonstrate that several OOD detection approaches, such as Baseline [29] and ODIN [30], have poor performance in detecting Knockoff Net attack samples. Hence, they propose a new OOD detection approach that leverages a classifier to detect OOD samples. However, the OOD detector is trained on samples that are from the same distribution used by the adversary to conduct Knockoff Net attacks, which is an unrealistic assumption in practice. SEAT [15] aims to detect model extraction attacks that use several similar samples to extract a target model, such as jacobian-based attacks [2], [3]. Hence, SEAT is ineffective when an adversary uses natural samples that are not similar to each other, such as Knockoff Net attack. VarDetect [14] uses Variational Autoencoders (VAs) and Maximum Mean Discrepancy (MMD) to detect model extraction attacks.

Recently, Jiang et al. [16] proposed a comprehensive defense that initially detects model extraction attack samples and then uses label filliping to reduce the surrogate model accuracy. The authors argue that classifiers are always more confident in classifying benign queries and less confident in classifying malicious queries. Thus, the maximum softmax probability (MSP) can be used as a metric to distinguish malicious queries. Since jacobian-based attacks also have large MSP, the authors increase the temperature parameter \( T \) in the Softmax function to mitigate the problem of classifier overconfidence. The proposed approach has two main drawbacks. First, It uses attack samples to determine defense parameters. Second, It only detects malicious samples and has no mechanism to detect adversaries. HODA can detect both jacobian-based and Knockoff Net attacks, and it performs well...
on high-dimensional datasets. Furthermore, unlike other work [11], [12], [16], [28], HODA does not need access to attack samples to determine its parameters.

C. Sample Hardness

The sample hardness has attracted attention from several machine learning domains, such as curriculum learning, identifying important or most informative examples, and detecting mislabeled samples [17], [31], [32]. We review only the most relevant work to our study. Hacohen et al. [18] and Mangalam and Prabhu [33] show that DNNs learn samples that are learnable by shallow models in early epochs of training before learning harder ones. The work of Hacohen et al. [18] that inspired our definition of hardness demonstrates that DNNs learn samples in both training and test sets in a similar order. Carlini et al. [31] propose several measures for identifying prototypical samples. They show prototypical samples are easy to learn, and training on hard samples can improve accuracy on many datasets and tasks. Toneva et al. [17] investigate the learning dynamics of neural networks and define a forgetting event to have occurred when a training sample transitions from being classified correctly to incorrectly over the course of learning. They show hard samples are forgotten with higher frequency than easy samples. Two works, [32] and [34], are particularly relevant to our work. Jiang et al. [32] introduce the consistency score that measures the structural consistency of an example with the underlying data distribution of target classifier’s training data. They demonstrate that easy samples have a higher consistency score, which means they lie in a region on the data manifold that is well supported by other regular instances. The authors examine several proxies for consistency score and indicate, in contrast to distance-based proxies, learning-speed-based proxies correlate very well with the consistency score. We demonstrate most model extraction attack samples are hard, which means that they are not well supported by the samples in the target model’s training data. In independent and concurrent work, Baldock et al. [34] introduce two measures for computing sample hardness, prediction depth and learning difficulty. The learning difficulty measure is the same as our measure of hardness, except that we use epoch rather than iteration. The authors demonstrate a strong correlation between prediction depth and learning difficulty. They show that samples learned in later epochs have higher prediction depth and confirm that neural networks learn easy samples first.

IV. HARDNESS DEGREE OF SAMPLES

We use the concept of learning difficulty [17], [18] to define our hardness measure. We expect that the predicted label of easy samples converges in early epochs and the predicted label of hard samples converges in the later epochs during training. The training process of a DNN-based classifier can be considered a sequence of subclassifiers so that each subclassifier is created at the end of an epoch. Suppose that classifier $f_i$ is trained for $m$ epochs. The training process of $f_i$ can be represented as the following sequence of subclassifiers:

$$F = (f_i^0, f_i^1, f_i^2, \ldots, f_i^{m-1})$$

where subclassifier $f_i^j$ is created at the end of the $i^{th}$ epoch. To compute the hardness degree of samples, we select a subsequence of $F$ called $F_{\text{subclf}}$. For example, if $m = 100$, $F_{\text{subclf}}$ can be $(f_1^{19}, f_1^{39}, f_1^{59}, f_1^{79}, f_1^{99})$. The hardness degree of sample $x_i$ is $h$ if the $h^{th}$ subclassifier in $F_{\text{subclf}}$ is the first subclassifier that the predicted label of all subsequent subclassifier in $F_{\text{subclf}}$ is equal to its predicted label. Therefore, the hardness degree of sample $x_i$ for classifier $f_i$, which is displayed by $\phi_{f_i}(x_i)$ is defined as follows:

$$\phi_{f_i}(x_i) = h$$

s.t. $\forall j > h, \overline{F_{\text{subclf}}[j]}(x_i) = \overline{F_{\text{subclf}}[j]}(x_i)$,

$$\overline{F_{\text{subclf}}[h]}(x_i) \neq \overline{F_{\text{subclf}}[h+1]}(x_i).$$

where $\overline{F_{\text{subclf}}[k]}(x_i)$ is the predicted label by $k^{th}$ subclassifier in $F_{\text{subclf}}$ for sample $x_i$. It is supposed that $\overline{F_{\text{subclf}}[-1]}(x_i) = \emptyset$. Based on the hardness degree definition, the hardness degree of a sample is in the range $[0, |F_{\text{subclf}}| - 1]$, where $|F_{\text{subclf}}|$ is the size of $F_{\text{subclf}}$. Since we want to calculate the hardness degree of samples at inference time for target model users’ samples, we need to save subclassifiers in $F_{\text{subclf}}$ during training to use them at inference time. When a new sample arrives, it is fed to all loaded subclassifiers in $F_{\text{subclf}}$, and using their predictions, the hardness degree of that sample is computed. It is important to note that we do not use the true label of samples to calculate their hardness degree.

We train three various types of classifiers, including DenseNet121 [35], ResNet18 [36], and MobileNet [37], on CIFAR10 and CIFAR100 training sets for 100 epochs. CIFAR10 dataset consists of 60K $32 \times 32 \times 3$ images in 10 classes. Each class has 5K training samples and 1K test samples. CIFAR100 dataset consists of 60K $32 \times 32 \times 3$ images in 100 classes. Each class has 500 training samples and
Based on their hardness degree, group $i$ samples of CIFAR10 and CIFAR100 datasets into ten groups. For easy samples, it is clear that they belong to their true label. For hard samples, it is not clear. The figure indicates that the easiest and hardest CIFAR10 test samples for each class to distinguish are shown. Figure 2 shows some examples of the easiest and hardest CIFAR10 test samples based on the hardness degree for each class.

For each range of hardness degrees, Data Percentage indicates the percentage of test samples whose hardness degrees are in that range. The results demonstrate that the misclassification rate is increased by increasing the hardness degree of samples. For example, the hardness degree of 40.65% of CIFAR10 test samples (4065 samples) is in the range [0, 9] for MobileNet classifier, from which 99.88% is classified correctly, or the hardness degree of 7.4% of CIFAR10 test samples (740 samples) is in the range [90, 99] for MobileNet classifier, from which 55.27% is classified correctly. More than 99% and 95% of CIFAR10 and CIFAR100 test samples with hardness degree $< 30$ are correctly classified. On the other side, less than 55% and 36% of CIFAR10 and CIFAR100 test samples with hardness degree $\geq 90$ are correctly classified.

Table II displays the Pearson correlation coefficients between hardness degree of CIFAR10 and CIFAR100 test samples for various pairs of classifiers. The results demonstrate a positive and strong correlation between the hardness degree of samples for various pairs of classifiers. The results indicate the hardness of samples is relatively transferable between different classifiers. As ResNet18 architecture achieves strong performance on both datasets at a reasonable computational cost, we use this architecture for target classifiers in the rest of the paper. We conduct various model extraction attacks on two CIFAR10 and CIFAR100 target classifiers in the next subsection to depict the hardness degree histogram of their samples.

### A. Model Extraction Attacks Setup

In line with prior work [10], [11], [12], we select JBDA [2], JBRAND [3], and Knockoff Net (K.Net) [4] model extraction attacks to evaluate our defense method. These attacks broadly represent two main strategies (synthesis or semantically similar samples) to conduct model extraction attacks. JBDA attack is implemented with $\lambda = 0.1$ and $\kappa = 2000$. The seed samples are selected from the test set of datasets. We use 500 (50 for each class) and 1000 (10 for each class) samples of CIFAR10 and CIFAR100 test sets for seed samples, respectively. For JBRAND attack, we generate three adversarial examples with random targets for each sample and use the same seed samples. For knockoff net attack, K.Net CIFARX, and K.Net TIN.

Fig. 1 shows the hardness degree histogram of CIFAR10 and CIFAR100 test samples for various pairs of classifiers. The figure demonstrates that the misclassification rate is increased by increasing the hardness degree of samples. For example, the hardness degree of 40.65% of CIFAR10 test samples (4065 samples) is in the range [0, 9] for MobileNet classifier, from which 99.88% is classified correctly, or the hardness degree of 7.4% of CIFAR10 test samples (740 samples) is in the range [90, 99] for MobileNet classifier, from which 55.27% is classified correctly. More than 99% and 95% of CIFAR10 and CIFAR100 test samples with hardness degree $< 30$ are correctly classified. On the other side, less than 55% and 36% of CIFAR10 and CIFAR100 test samples with hardness degree $\geq 90$ are correctly classified.

Table II displays the Pearson correlation coefficients between hardness degree of CIFAR10 and CIFAR100 test samples for various pairs of classifiers. The results demonstrate a positive and strong correlation between the hardness degree of samples for various pairs of classifiers. The results indicate the hardness of samples is relatively transferable between different classifiers. As ResNet18 architecture achieves strong performance on both datasets at a reasonable computational cost, we use this architecture for target classifiers in the rest of the paper. We conduct various model extraction attacks on two CIFAR10 and CIFAR100 target classifiers in the next subsection to depict the hardness degree histogram of their samples.

100 test samples [38]. All classifiers are trained using stochastic gradient descent with momentum 0.9 and batch size 128. The learning rate is 0.1 and it is scheduled to be decreased in each epoch by a constant factor of 0.955. The accuracy of classifiers is presented in Table I. We save all 100 subclassifiers in the training phase of each classifier and use them to calculate the hardness degree of samples ($|F_{\text{subcl}}| = 100$). Figure 1 shows the hardness degree histogram of CIFAR10 and CIFAR100 test samples for various classifiers. The figure demonstrates that a large fraction of CIFAR10 test samples is easy, and CIFAR100 test set has more number of hard samples than CIFAR10 test set. Figure 2 shows some examples of the easiest and hardest CIFAR10 test samples for each class to sanity check our hardness measure. The figure indicates that for easy samples, it is clear that they belong to their true label class, but for hard samples, it is not clear.

To assess the relationship between the hardness degree of samples and the misclassification rate, we partition test samples of CIFAR10 and CIFAR100 datasets into ten groups based on their hardness degree. Group $i$ consists of samples that their hardness degree is in the range $[i \times 10, (i+1) \times 10]$. Hence, the first hardness group consists of the easiest samples, and the last hardness group consists of the hardest ones. It is important to note that the number of samples in each group is different. Figure 3 indicates the relation between the hardness degree of CIFAR10 and CIFAR100 test samples and the accuracy and the misclassification rate of various classifiers. Also, it indicates the percentage of samples in each hardness degree range by a green curve. The figure demonstrates that the misclassification rate is increased by increasing the hardness degree of samples. For example, the hardness degree of 40.65% of CIFAR10 test samples (4065 samples) is in the range [0, 9] for MobileNet classifier, from which 99.88% is classified correctly, or the hardness degree of 7.4% of CIFAR10 test samples (740 samples) is in the range [90, 99] for MobileNet classifier, from which 55.27% is classified correctly. More than 99% and 95% of CIFAR10 and CIFAR100 test samples with hardness degree $< 30$ are correctly classified. On the other side, less than 55% and 36% of CIFAR10 and CIFAR100 test samples with hardness degree $\geq 90$ are correctly classified.

### Table II: Pearson Correlation Coefficients Between Hardness Degree of CIFAR10 and CIFAR100 Test Samples for Various Pairs of Classifiers

| Classifier Pairs | Pearson Correlation Coefficient |
|------------------|---------------------------------|
| CIFAR10          | CIFAR100                        |
| ResNet18-DenseNet | 0.775                           |
| ResNet18-MobileNet | 0.765                           |
| DenseNet121-MobileNet | 0.769                           |

Fig. 2 shows some examples of the easiest and hardest CIFAR10 test samples based on the hardness degree for each class. The learning rate is 0.1 and it is scheduled to be decreased in each epoch by a constant factor of 0.955. The accuracy of classifiers is presented in Table I. We save all 100 subclassifiers in the training phase of each classifier and use them to calculate the hardness degree of samples ($|F_{\text{subcl}}| = 100$). Figure 1 shows the hardness degree histogram of CIFAR10 and CIFAR100 test samples for various classifiers. The figure demonstrates that a large fraction of CIFAR10 test samples is easy, and CIFAR100 test set has more number of hard samples than CIFAR10 test set. Figure 2 shows some examples of the easiest and hardest CIFAR10 test samples for each class to sanity check our hardness measure. The figure indicates that for easy samples, it is clear that they belong to their true label class, but for hard samples, it is not clear.

To assess the relationship between the hardness degree of samples and the misclassification rate, we partition test samples of CIFAR10 and CIFAR100 datasets into ten groups based on their hardness degree. Group $i$ consists of samples that their hardness degree is in the range $[i \times 10, (i+1) \times 10]$. Hence, the first hardness group consists of the easiest samples, and the last hardness group consists of the hardest ones. It is important to note that the number of samples in each group is different. Figure 3 indicates the relation between the hardness degree of CIFAR10 and CIFAR100 test samples and the accuracy and the misclassification rate of various classifiers. Also, it indicates the percentage of samples in each hardness degree range by a green curve. The figure demonstrates that the misclassification rate is increased by increasing the hardness degree of samples. For example, the hardness degree of 40.65% of CIFAR10 test samples (4065 samples) is in the range [0, 9] for MobileNet classifier, from which 99.88% is classified correctly, or the hardness degree of 7.4% of CIFAR10 test samples (740 samples) is in the range [90, 99] for MobileNet classifier, from which 55.27% is classified correctly. More than 99% and 95% of CIFAR10 and CIFAR100 test samples with hardness degree $< 30$ are correctly classified. On the other side, less than 55% and 36% of CIFAR10 and CIFAR100 test samples with hardness degree $\geq 90$ are correctly classified.

Table II displays the Pearson correlation coefficients between hardness degree of CIFAR10 and CIFAR100 test samples for various pairs of classifiers. The results demonstrate a positive and strong correlation between the hardness degree of samples for various pairs of classifiers. The results indicate the hardness of samples is relatively transferable between different classifiers. As ResNet18 architecture achieves strong performance on both datasets at a reasonable computational cost, we use this architecture for target classifiers in the rest of the paper. We conduct various model extraction attacks on two CIFAR10 and CIFAR100 target classifiers in the next subsection to depict the hardness degree histogram of their samples.

### A. Model Extraction Attacks Setup

In line with prior work [10], [11], [12], we select JBDA [2], JBRAND [3], and Knockoff Net (K.Net) [4] model extraction attacks to evaluate our defense method. These attacks broadly represent two main strategies (synthesis or semantically similar samples) to conduct model extraction attacks. JBDA attack is implemented with $\lambda = 0.1$ and $\kappa = 2000$. The seed samples are selected from the test set of datasets. We use 500 (50 for each class) and 1000 (10 for each class) samples of CIFAR10 and CIFAR100 test sets for seed samples, respectively. For JBRAND attack, we generate three adversarial examples with random targets for each sample and use the same seed samples as JBDA. Each sample is perturbed in five iterations with $\epsilon = \frac{64}{225 \times 255}$, $\lambda = \frac{64}{225}$, and $\kappa = 2000$. We consider two versions of knockoff net attack, K.Net CIFARX, and K.Net TIN.
TABLE III

| Model Extraction Attack | CIFAR10 Acc. (%) | CIFAR10 Fid. (%) | CIFAR100 Acc. (%) | CIFAR100 Fid. (%) |
|-------------------------|------------------|-----------------|------------------|------------------|
| Prob. Vec.              | 45.33            | 66.88           | 81.36            | 72.43            |
| Label                   | 79.86            | 80.18           | 81.36            | 72.43            |
| Prob. Vec.              | 45.33            | 66.88           | 81.36            | 72.43            |
| Label                   | 79.86            | 80.18           | 81.36            | 72.43            |

K.Net CIFARX attack uses CIFAR100 training set to extract CIFAR10 target classifier and vice versa. K.Net TIN employs TinyImageNet [39] training set to extract target classifiers. TinyImageNet dataset is a subset of ILSVRC12 dataset [26], and contains 200 classes. It has 500 training samples and 50 test samples for each class. The size of images is $64 \times 64$. We resize all images to $32 \times 32$. We use random strategy to implement knockoff net attacks. To evaluate the performance of model extraction attacks, we use two ResNet18 classifiers being trained on CIFAR10 and CIFAR100 training sets as the target classifiers and conduct all four attacks on them. The attack budget in our experiments is $B=50000$ (same as [10], [11], [12]). Table III shows the accuracy and the fidelity of surrogate classifiers created by various model extraction attacks on CIFAR10 and CIFAR100 test samples. The results demonstrate that K.Net attacks have significantly better performance than jacobian-based attacks (JBDA and JBRAND), and when the output of target classifiers is probability vector, the performance of attacks is considerably increased.

B. Hardness of Model Extraction Attack Samples

Figure 4 depicts the hardness degree histogram of 50000 samples generated by various attacks for CIFAR10 and CIFAR100 target classifiers. In this experiment, the architecture of target classifiers is ResNet18. We also present the hardness degree histogram of attack samples when the architecture of target classifiers is DenseNet121 in Figure 5.

Both figures demonstrate that the samples generated by various attacks have a very small number of easy samples, and most samples have medium or high hardness degrees. Figure 6 displays two-dimensional visualization of CIFAR10 test samples using t-SNE [40]. Figure 6a uses the logits of the CIFAR10 classifier to visualize CIFAR10 test samples, and the color of each sample is determined by its label. This figure has ten sample clusters where most samples of each cluster belong to a class. Figure 6b illustrates the hardness degree of CIFAR10 test samples for CIFAR100 target classifier and demonstrates that most of the easy samples are in the high-density regions inside clusters, and most of the hard samples are in the low-density regions at the borders of clusters. Figure 6c is similar to Figure 6b, but the hardness degree of each sample is calculated via CIFAR100 target classifier. This figure demonstrates when the training data distribution of the classifier being used to calculate the hardness degree of samples becomes different from the distribution of CIFAR10 test samples, the hardness degree of a high number of samples is changed. Figure 6c shows hard and medium samples are distributed among clusters, and the number of easy samples is decreased.
very small. Similar to Figure 6, we visualize CIFAR100 test samples and their hardness for CIFAR10 and CIFAR100 target classifiers in Figure 7. The experiments demonstrate that the hardness degree of a sample for a classifier pertains to the training data distribution of that classifier.

Based on our experiments and the findings of previous work [31], [32], easy samples lie on the data manifold that is well supported by the target classifier training data. In other words, since easy samples have patterns that are prevalent among training samples, the target model trains several times on those patterns in one epoch. Hence, the target model needs fewer epochs to learn those patterns, and thereby, the predicted label of easy samples converges in early epochs. On the other side, since hard samples have rare patterns, the target model needs more epochs to learn those patterns, and thereby, the predicted label of hard samples converges in later epochs. Figures 4 and 8 demonstrate that, unlike normal samples, the number of easy samples among model extraction attack samples is very small, which means that attack samples do not lie on the data manifold that is well supported by the target classifier training data. In other words, since attack samples have patterns that are not prevalent among training samples of the target model, their predicted labels do not converge in early epochs. For simplicity, we write histogram rather than hardness degree histogram in the rest of the paper.

C. HODA: Hardness-Oriented Detection Approach

We propose Hardness-Oriented Detection Approach (HODA) to detect sample sequences of model extraction attacks in order to increase the cost of attacks. We assume that each user of the target classifier has an account and all histograms in Hist Set are independent (i.i.d.) from the target classifier’s training data distribution. \( H_u \) serves as a benchmark, providing an expected hardness degree histogram based on standard samples. Once the number of samples provided by user \( u \) reaches a specific threshold, defined as \( num_u \), HODA calculates the Pearson distance between the user’s histogram \( H_u \) and the reference histogram \( H_n \). If this calculated distance surpasses a pre-set threshold \( \delta \), then user \( u \) is flagged as a potential adversary. The Pearson distance (PD) between two random variables \( X \) and \( Y \) is defined as follows:

\[
PD(X, Y) = 1 - \frac{\text{Cov}(X, Y)}{\rho_X \rho_Y} \tag{6}
\]

where \( \text{Cov}(X, Y) \) is the covariance between random variables \( X \) and \( Y \), and \( \rho_X \) is the standard deviation of random variable \( X \). The output of Pearson distance is in the range \([0,2]\). The output of Pearson distance indicates the inconsistency between samples sent by user \( u \) and normal samples. To calculate the Pearson distance between two histograms, HODA first transforms histograms into probability vectors by dividing the value of histogram bins by the total number of samples in the histogram (\( H_u/\text{sum}(H_u) \) and \( H_n/\text{sum}(H_n) \)) and then, calculates the Pearson distance between them. HODA uses normal sample set \( S_{\text{HODA}} \) to create \( H_n \) and calculate \( \delta \). It randomly selects \( num_{\text{seq}} \) sample sequences with size \( num_s \) from the sample set \( S_{\text{HODA}} \) and for each sample sequence, produces a histogram and adds it to the histogram set \( \text{Hist Set} \). The normal histogram \( H_n \) is the average of all histograms in \( \text{Hist Set} \), and \( \delta \) is the maximum Pearson distance between \( H_n \) and all histograms in \( \text{Hist Set} \). Since \( \delta \) is independent of attacks and only relies on normal samples, HODA is not dependent on any attacks. Notably, HODA does not need to save samples of each user or their hardness degrees. It only keeps a vector (\( H_u \)) that indicates the values of histogram bins for each user. Algorithm 1 describes HODA in details.

Algorithm 1 Hardness-Oriented Detection Approach (HODA)

**Inputs:** \( S_{\text{HODA}} \) is a set of normal samples, \( num_u \) is the size of sample sequences, \( num_{\text{seq}} \) is the number of sample sequences. \( F_{\text{subclf}} \) is the subclassifier subsequence, \( \text{NewQuery} \) is the newest query being received by the target model, and \( \text{UserID} \) is the owner identifier of \( \text{NewQuery} \).

**Outputs:** \( H_n \) is the histogram of normal samples, \( \delta \) is the attack detection threshold, \( \text{AttackAlarm} \) declares the occurrence of attack.

1. function GETHARDNESSDEGREE(x, F_{\text{subclf}})  
2. \quad label ← None  
3. \quad for i ← 0:len(F_{\text{subclf}}) do  
4. \quad \quad pred_vector ← F_{\text{subclf}}(i)(x) // F_{\text{subclf}}(i) is the i^{th} subclassifier in F_{\text{subclf}}  
5. \quad \quad degree ← i  
6. \quad \quad pred_label ← argmax(pred_vector)  
7. \quad \quad if pred_label ≠ label then  
8. \quad \quad \quad degree ← i  
9. \quad \quad label ← pred_label  
10. \quad end if  
11. \quad end for  
12. \quad return degree  
13. end function  
14. function PEARSONDIST(H_n, H_u)  
15. \quad return \text{PD}(H_n/\text{sum}(H_n), H_u/\text{sum}(H_u))  
16. end function  
17. function HODAINITIALIZATION(S_{\text{HODA}}, num_s, num_{\text{seq}}, F_{\text{subclf}})  
18. \quad Hist Set ← ∅  
19. \quad for i ← 0, num_{\text{seq}} do  
20. \quad \quad seq ← Randomly select num_s samples from \( S_{\text{HODA}} \)  
21. \quad \quad Hist Set ← Hist Set ∪ Hist  
22. \quad \quad for s in seq do  
23. \quad \quad \quad Hist = GetHardnessDegree(s, F_{\text{subclf}})  
24. \quad \quad \quad Hist[Hist] += 1  
25. \quad \quad end for  
26. \quad \quad Hist = Hist Set ∪ Hist  
27. \quad \quad Dist List ← ∅  
28. \quad \quad for Hist in Hist Set do  
29. \quad \quad \quad Hist List.append(PEARSONDIST(H_n, Hist))  
30. \quad \quad end for  
31. \quad \quad δ ← Max(Dist List)  
32. \quad \quad return H_n, δ  
33. end function  
34. function HODA(NewQuery, UserID, H_n, δ, num_s, F_{\text{subclf}})  
35. \quad \text{AttackAlarm} ← False  
36. \quad \quad H_u ← GetUserHistogram(UserID)  
37. \quad \quad HD ← GetHardnessDegree(NewQuery, F_{\text{subclf}})  
38. \quad \quad HD[HD] += 1  
39. \quad \quad if Sum(HD) = num_s then  
40. \quad \quad \quad if PEARSONDIST(H_n, HD) > δ then  
41. \quad \quad \quad \quad AttackAlarm ← True  
42. \quad \quad \quad end if  
43. \quad \quad \quad end if  
44. \quad \quad H_u ← Avg(Hist Set)  
45. \quad \quad SaveUserHistogram(H_u, UserID)  
46. \quad \quad return AttackAlarm  
47. \quad \endfunction  
48. end function
is the only parameter of PRADA. Since PRADA needs to save each user’s samples and calculate \( L_2 \) distance between them, it has a high computational overhead. VarDetect [14] computes the Maximum Mean Discrepancy (MMD) distance between distributions of user samples and normal samples in the latent space of a variational autoencoder, and if the distance exceeds threshold \( \delta \), the user is detected as an adversary. We follow the default configurations of VarDetect. However, since VarDetect has no mechanism to compute \( \delta \), we use the mechanism of HODA for computing \( \delta \) to determine \( \delta \) for VarDetect.

The proposed method by Jiang et al. [16] is not directly comparable with HODA. The authors use the maximum soft-max probability along with temperature-scaling (TMSP) to detect a single malicious sample. However, HODA, PRADA, and VarDetect aim to detect malicious users, not malicious queries. To demonstrate that hardness degree is a better metric than TMSP to detect malicious users, we consider a new defense called TMSPDetect. TMSPDetect is exactly the same as HODA, but it uses TMSP rather than hardness degree to create histograms. In [16], attack and benign samples are used to determine the temperature parameter \( T \). However, in HODA threat model, it is assumed that the defender has no access to the attack samples. Despite that the chosen \( T \) in [16] is based on attack samples, we choose the average \( T \) reported in [16], \( T = 1.5 \), for computing TMSP in TMSPDetect. Also, we use 100 bins to create TMSP histograms.

Table IV indicates the detection rate, AUC score, and False Positive Rate (FPR) of PRADA, VarDetect, TMSPDetect, HODA-5, and HODA-11 against four various model extraction attacks on CIFAR10 and CIFAR100 target classifiers. We evaluate defenses with various values for \( num_s \) to better compare their capabilities. We consider \( num_s = 100 \) as baseline. All defenses have very low false-positive rates. False-Positive Rate (FPR) indicates the percentage of benign users’ samples sequences wrongly detected as an attack. AUC score is independent of \( \delta \) and indicates the defense performance to separate adversaries from benign users. The results demonstrate that HODA is very effective against model extraction attacks, and HODA-11 is slightly better than HODA-5. HODA outperforms PRADA and VarDetect by a large margin. The performance of TMSPDetect is comparable with HODA against Knockoff Net attacks. However, HODA has a better detection rate and AUC score against Jacobian-based attacks.

### A. Computational Cost Analysis

Table V compares PRADA, VarDetect, TMSPDetect, HODA-11, and HODA-5 in terms of the average runtime, memory consumption, and number of predictions for each sample. The runtime is measured for 500 sequences of attack samples and indicates the efficiency of the defenses. PRADA is the slowest due to its complex mechanism of calculating the hardness degree. VarDetect is slightly faster than PRADA but requires more memory. TMSPDetect is the fastest and consumes less memory than VarDetect. However, HODA-11 and HODA-5 have comparable runtime and memory consumption but are 10 times faster than VarDetect.

### V. Setup and Evaluation

Two normal sample sets \( S_{HODA} \) and \( S_s \) are required to evaluate the performance of HODA. \( S_s \) is used to simulate benign users. We randomly select 40% and 60% of test samples of each dataset for \( S_{HODA} \) and \( S_s \), respectively. We randomly select \( num_{seq} = 40000 \) sequences with size \( num_s \) from \( S_{HODA} \) to create \( H_d \) and calculate \( \delta \). To evaluate the performance of HODA against model extraction attacks, we simulate 10000 benign users and 10000 adversaries for each attack. Each benign user sends a sequence of \( num_s \) samples randomly selected from \( S_s \), and each adversary sends a sequence of \( num_s \) samples randomly selected from 50000 samples of attack in the order they were generated. So far, we have used 100 subclassifiers to calculate the hardness degree of samples. However, it may not be possible to classify each sample by a high number of subclassifiers in practice. So in order to reduce the computational cost of HODA, we consider two versions of HODA called HODA-11 and HODA-5. HODA-11 uses 11 subclassifiers \( F_{subclf} = \{ f^0, f^9, f^{19}, f^{29}, f^{39}, f^{49}, f^{59}, f^{69}, f^{79}, f^{89}, f^{99} \} \), and HODA-5 uses 5 subclassifiers \( F_{subclf} = \{ f^0, f^9, f^{19}, f^{29}, f^{39} \} \) to calculate the hardness degree of each sample. Since the hardness degree domain depends on the number of subclassifiers in \( F_{subclf} \), the hardness degree in HODA-11 and HODA-5 is in the ranges \([0,10]\) and \([0,4]\), respectively.

We compare HODA with PRADA [3], VarDetect [14], and TMSPDetect [16]. PRADA [3] declares that the histogram of minimum \( L_2 \) distance between a new sample and all previous samples of a benign user follows a Gaussian distribution. Hence, it uses the Shapiro-Wilk normality test to determine that a sample sequence belongs to a benign user or an adversary. Similar to HODA, PRADA also uses threshold \( \delta \) to detect sample sequences of model extraction attacks, and \( \delta \) is the only parameter of PRADA. Since PRADA needs to save each user’s samples and calculate \( L_2 \) distance between them, it has a high computational overhead. VarDetect [14] computes the Maximum Mean Discrepancy (MMD) distance between distributions of user samples and normal samples in the latent space of a variational autoencoder, and if the distance exceeds threshold \( \delta \), the user is detected as an adversary. We follow the default configurations of VarDetect. However, since VarDetect has no mechanism to compute \( \delta \), we use the mechanism of HODA for computing \( \delta \) to determine \( \delta \) for VarDetect.
average memory consumption, and the number of model predictions to defend CIFAR10 target classifier in \( n_{\text{m}} = 500 \) on Tesla K80 GPU. Since HODA does not need to store samples in input space or latent space, the memory consumption of HODA is significantly less than PRADA and VarDetect for each user. HODA needs to store a vector of integers for each user, representing the hardness degree histogram. TMSPDetect needs to store a vector of float numbers for each user, representing the TMSP histogram. The runtime of HODA and TMSPDetect is several times smaller than PRADA and VarDetect. Besides, the runtime of PRADA and VarDetect increases by increasing \( n_{\text{m}} \), but the runtime of HODA and TMSPDetect is constant for any \( n_{\text{m}} \).

While HODA stores less data per user compared to TMSPDetect, it is required to maintain 5 or 11 subclassifiers for use during inference. For instance, the size of DenseNet121, our largest model in the experiments, is approximately 30 megabytes, and HODA-11 requires 330 megabytes to store these subclassifiers. In contrast, TMSPDetect only needs to store a single model. To capitalize on HODA's memory efficiency advantage over TMSPDetect, the service employing HODA for defending against adversaries should have a large user number. It's worth noting that since TMSPDetect utilizes HODA's mechanism for detecting adversaries, it inherits HODA's efficiency. Although HODA-5 (HODA-11) requires the predictions of 5 (11) models to calculate the hardness degree of each sample, there is no sequential relationship between models, and they can predict in parallel, so HODA does not increase the inference time of target models. Overall, the main drawback of HODA is in the number of predictions that can be solved by providing several GPUs. On the other side, memory consumption and runtime of VarDetect and PRADA are extremely more than HODA for each user. Hence, HODA is more scalable than PRADA and VarDetect with respect to the number of users.

B. Transfer Learning on High-Dimensional Datasets

Transfer learning is a technique that initializes the parameters of the target task classifier using the parameters of a pre-trained source task classifier. To demonstrate the generalizability of HODA on high-dimensional datasets and across various DNN architectures, we train six new target classifiers with various architectures on CUB200 and Caltech256 datasets using transfer learning. We use DenseNet121 [35], ResNet18 [36], and MobileNet [37] architectures to create target models. CUB200 dataset [41] contains 200 classes of bird categories, and it has about 6K training and about 6K test samples. Caltech256 dataset [42] contains 256 classes of common objects categories, and it has about 24K training and about 6K test samples. The size of both dataset images is \( 224 \times 224 \times 3 \). The training process of new target classifiers is the same as CIFAR10 and CIFAR100 target classifiers (Section IV). We initialize the parameters of target classifiers from a pre-trained ImageNet [26] classifier and train all layers of target classifiers. Orekondy et al. [10] indicate that jacobian-based model extraction attacks have very poor performance on high dimensional datasets. Thereby, we only evaluate the performance of HODA against K.Net ILSVRC12 attack. K.Net ILSVRC12 is the Knockoff Net attack that uses ILSVRC12 dataset as the surrogate classifier’s training set. ILSVRC12 dataset uses a subset of ImageNet dataset [26] in which there exist about 1.2 million training images, 50K validation images, and 100K test images. ILSVRC12 dataset has 1000 classes and the size of images is \( 224 \times 224 \times 3 \). The budget of K.Net ILSVRC12 is 50000, and the output of target classifiers is the entire probability vector. The accuracy of ResNet18 CUB200 target classifier and its surrogate classifier is 73.7% and 59.3%, respectively, and the accuracy of ResNet18 Caltech256 target classifier and its surrogate classifier is 77.2% and 72.2%, respectively.

Figure 8 depicts the hardness degree histogram of CUB200 and Caltech256 test sets on the associated ResNet18 target classifier and also, the hardness degree histogram of K.Net ILSVRC12 samples for both target classifiers. We employed 100 subclassifiers to calculate the hardness degree in Figure 8 in order to clearly illustrate the distinction between hardness degree histograms for normal and attack samples. However, for our simulation presented in Table VI, we utilized 5 (HODA-5) or 11 (HODA-11) subclassifiers to determine the hardness degree of the samples. The figure demonstrates that the majority number of K.Net ILSVRC12 attack samples are hard, and the number of easy samples is very small. We replicate the experiments of Section V to evaluate the performance of HODA against K.Net ILSVRC12 attack with the same parameters for three different target classifier architectures, including ResNet18, DenseNet121, and MobileNet. Table VI shows the performance of TMSPDetect, HODA-11 and HODA-5 against K.Net ILSVRC12 attack on various target classifiers. The results demonstrate that although the starting point of target classifiers’ parameters is not random, HODA is very effective in detecting K.Net ILSVRC12 attack. The detection rate, AUC score, and false positive rate of HODA are almost the same among various target classifier architectures, demonstrating the generalizability of HODA across different architectures. The detection rate and AUC score of TMSPDetect are lower than HODA, which indicates that the hardness degree is a better metric than TMSP to detect malicious users on high-dimensional datasets. TMSPDetect should monitor almost two times more samples to detect model extraction attacks with a high success rate.

VI. DISCUSSION ON ADAPTIVE ADVERSARY

An adaptive adversary who is aware of HODA must send her queries based on the hardness degree histogram of normal
samples to evade HODA. We consider three scenarios for an adaptive adversary to conduct model extraction attacks. In the first scenario, the adversary has no access to normal samples, and it can only use synthetic or semantically similar samples to extract the target model. There are two reasons why such attacks are hard to conduct. First, the adversary needs samples with various degrees of hardness, but since the adversary has no access to the target classifier, it cannot determine the hardness degree of her samples for the target classifier. Second, the adversary has no access to the histogram of normal samples to generate her samples based on it.

In the second scenario, we assume the adversary has access to a limited number of normal samples, and it can use normal samples to make her hardness degree histogram more similar to the hardness degree histogram of normal samples. To evaluate HODA in this scenario, we suppose that the adversary has access to 1000 normal samples from $S_u$ and it sends a sample sequence of which $P_n$% is filled by normal samples, and the rest is filled by model extraction attack samples. Notably, when the number of normal samples in the sequence exceeds 1000, the adversary sends duplicate normal samples. Using this observation, we proposed Hardness-Oriented Detection Approach (HODA) to detect model extraction attacks. HODA can detect malicious accounts by monitoring 100 (or 50) samples of model extraction attacks. Hence, an adversary must create multiple user accounts to send model extraction samples. In this scenario, the adversary must create $\frac{5000}{50} = 100$ (or $\frac{5000}{100} = 50$) user accounts to conduct model extraction attacks. HODA can detect the sample sequences of model extraction attacks. In tasks that have significant economic benefits for model owners or are security-sensitive, it is reasonable to suppose that creating an account for the target model’s users is not priceless or without any monitoring (phone number or email). In such situations, HODA increases the attack cost up to 500 (or 1000) times with respect to the number of user accounts that an adversary must create.

### VII. Conclusion

This paper demonstrates that the hardness degree of samples is important in trustworthy machine learning. We investigated the hardness degree of samples and demonstrated that the hardness degree histogram of model extraction attack samples is different from the hardness degree histogram of normal samples. Using this observation, we proposed Hardness-Oriented Detection Approach (HODA) to detect sample sequences of model extraction attacks. HODA can detect the sample sequences of model extraction attacks with a high success rate, and it outperforms all previous model extraction detection methods.

### REFERENCES

[1] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction APIs,” in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 601–618.

[2] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 506–519.

[3] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, “High accuracy and high fidelity extraction of neural networks,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4949–4958.

[4] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, “High accuracy and high fidelity extraction of neural networks,” in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1345–1362.
