Model Stealing Attacks On FHE-based Privacy-Preserving Machine Learning through Adversarial Examples

Bhuvnesh Chaturvedi[1], Anirban Chakraborty[1], Ayantika Chatterjee[1], and Debdeep Mukhopadhyay[1]

Indian Institute of Technology Kharagpur
{bhuvneshchaturvedi2512,ch.anirban00727,cayantika,debdeep.mukhopadhyay}@gmail.com

Abstract. Classic MLaaS solutions suffer from privacy-related risks since the user is required to send unencrypted data to the server hosting the MLaaS. To alleviate this problem, a thriving line of research has emerged called Privacy-Preserving Machine Learning (PPML) or secure MLaaS solutions that use cryptographic techniques to preserve the privacy of both the input of the client and the output of the server. However, these implementations do not take into consideration the possibility of transferability of known attacks in classic MLaaS settings to PPML settings. In this paper, we demonstrate that it is possible to transfer existing model-extraction attacks using adversarial examples to PPML settings due to relaxed constraints on the abilities of the adversary. We show a working example of an end-to-end attack on an image processing application built using a popular FHE-based framework, namely Concrete-ML. We successfully create a cloned model with just 5000 queries, which is, in fact, 10× less than the size of the training set of the victim model, while incurring only a 7% loss in accuracy. Further, we incorporate the well-known defense strategy against such attacks and show that our attack is still able to clone the model. Finally, we evaluate the different defense rationales that exist in literature and observe that such model stealing attacks are difficult to prevent in secure MLaaS settings.

Keywords: Fully Homomorphic Encryption · Privacy Preserving Machine Learning · MLaaS · Adversarial Examples · Model Stealing Attack

1 Introduction

In recent decades, the advent of machine learning (ML) has heralded a paradigm shift in computational methodologies, revolutionizing the way we approach complex tasks across a myriad of domains. Rooted in the field of artificial intelligence, machine learning endows computer systems with the capacity to learn and adapt from data, enabling them to make intelligent decisions or predictions without being explicitly programmed. This transformative capability has found widespread applications in diverse sectors including healthcare [36], finance [20], natural language processing [45], computer vision [40], and more. The proficiency of machine learning algorithms to discern intricate patterns within vast datasets has

not only propelled the advancement of technology but has also opened avenues for unprecedented insights and solutions. As the boundaries of its potential continue to expand, the integration of ML techniques promises to be an instrumental force in shaping the future of scientific inquiry and technological innovation.

Machine learning often involves training complex models such as Convolution Neural Network (CNN) [2] and Deep Neural Networks (DNN) [48]. However, training such models requires collecting a significant amount of labeled data that is not readily available and takes a lot of effort and resources to collect. Moreover, processing such raw data consumes a lot of time and money and additionally requires manpower to correctly annotate the collected data. Additionally, running complex models requires a lot of computational power, which necessitates procuring and maintaining costly hardware and associated infrastructure. To alleviate this problem, a new thriving field has emerged in recent years where big organizations with sufficient resources, both in terms of hardware capability as well as huge volumes of real-world data, train their proprietary ML models and then offer access to the model as a service to their clients for use. This practice is referred to as Machine Learning as a Service (MLaaS), where the clients or users are only given access to the prediction or classification APIs that they use to query the model on their own inputs.[1]

In existing MLaaS (which we will refer to as classic MLaaS going forward), the client queries the model on clear data and also obtains the result in the clear. This setting gives rise to privacy issues since the data used to query the model is often sensitive and involves various privacy concerns. In many cases, even the prediction output from the MLaaS model is required to be kept confidential and only accessible to authorized parties. For example, a hospital using a pay-per-use MLaaS model is required to uphold the privacy and anonymity of individual patients' data as well as the analytics results from the model. To address this problem, a new class of MLaaS has emerged in recent times, which is referred to as Privacy Preserving Machine Learning (PPML) or *Secure MLaaS*. In these schemes, both the data used to query the model and its predictive output are encrypted using the client's secret key such that no information is leaked to the server. In other words, the server is considered *honest-but-curious* wherein it is expected to perform the required computations judiciously while making attempts to learn passive information about the client's data.

The majority of such secure MLaaS schemes use different cryptographic primitives such as Secure Multiparty Computations (SMPC) [24], Fully Homomorphic Encryption (FHE) [38], or a hybrid of both [21] to ensure the confidentiality of both client's sensitive data as well as prediction results from the server. SMPC-only and hybrid solutions require the active participation of clients in the protocol, which increases communication costs between the server hosting the model and the client. On the other hand, FHE-only solutions are closer to the classic MLaaS setting as the client sends its encrypted inputs to the server and only receives the final (encrypted) prediction results as the output, which can

[1] On a commercial scale, the model owner charges the user for each query they make, which is either paid per query or paid in total at the end of a billing cycle.

be decrypted only by the client to obtain the final result. Nonetheless, all these solutions guarantee to protect the privacy of the client's data from a malicious server. However, while these schemes inherently assume the client to be an honest user of the service, in practice, the client may itself be actively malicious and might undertake activities to leak private information from the server. One such activity is *model stealing attack*, where the client aims to prepare a clone of the server's model by using its prediction services, thereby resulting in the theft of intellectual property.

Since the collection and labeling of training data incur a huge cost, it is often considered proprietary information and thus is not made public by the data owner. Moreover, the training data may also contain sensitive information, such as financial or medical records, which are required to be kept confidential. This restricts anyone but the data owner from training their own model. Therefore, model stealing attacks are practical threats to MLaaS providers with financial implications. An adversary can simply steal the model at a cost significantly less than the one spent by the model owner. Once obtained, the adversary is free to either keep the model for its own usage without paying any fee or may host it as its own service. While this is detrimental to the business of the original model owner, the adversary can use the stolen model to carry out further attacks such as membership inference attacks [28] which involves inferring whether a sample point belongs to the training data or not, or to generate adversarial examples [51] to force misclassification by the original model.

1.1 Motivation

While model stealing attacks using adversarial examples have been proposed in literature [13, 52], the threat model and underlying assumptions of such works do not directly apply to secure MLaaS settings. For example, in the context of computer vision, adversarial examples are generated by adding perturbations to the original images while ensuring that these perturbations remain imperceptible to the human eye.[2] However, in the context of secure MLaaS, the model operates over encrypted data in order to preserve the privacy of the client inputs. The underlying encryption schemes guarantee that the ciphertexts received by the model provider do not leak any information about the underlying plaintext. A side-effect of this guarantee is that the ciphertexts also hide the information about any perturbations being added to the original inputs. In other words, from the perspective of the server, the encryption of the original image looks similar to the encryption of the perturbed image. Thus, unlike classic MLaaS, where the adversary needs to carefully add perturbations to an original image, in secure MLaaS, the adversary is free to add random perturbations while remaining undetected.

The majority of existing defense mechanisms against model-stealing attacks depend on the ability to detect adversarial inputs [22,29,33,35]. While a number of such defenses are available for classic MLaaS applications, *to the best of our*

[2] In the context of MLaaS, this human can be a system administrator of the server on which the model is running.

knowledge, the applicability and efficacy of these defense mechanisms have not been evaluated in the context of secure MLaaS. Most of these defenses depend on operations over the input features, which are encrypted in secure MLaaS setting. Moreover, the output of these operations also remains encrypted and can only be decrypted by the secret key which is unavailable to the server.[3] This motivates us to examine the existing defenses and their capability to defend against model-stealing attacks on secure MLaaS.

1.2 Contributions

In this paper, we first highlight the existing methods of generating adversarial examples and discuss how the constraints that are required to generate these examples are not applicable to secure MLaaS. We then show a demonstration of a model stealing attack on a simple Fully Connected Network (FCN) built using a well-known FHE-only framework, where the accuracy of our stolen model turned out to be 89% while requiring only 4000 queries to the victim model. It is worth mentioning that we focus on an FHE-only framework for secure MLaaS as these schemes provide end-to-end encryption and do not require interference from the client. Alternative schemes like SMPC-only solutions or hybrid ones are costly in terms of communication rounds and data exchanges. In addition, hybrid solutions are themselves known to be vulnerable to model stealing attacks [7,26]. Moreover, FHE-only solutions are often proposed as potential countermeasures against attacks on hybrid solutions [7]. Finally, we evaluate the existing defenses against model stealing using adversarial examples and highlight their relevance in the context of secure MLaaS.

1.3 Organization

The rest of the paper is organized as follows: Section 2 provides a brief background of the concepts of machine learning, fully homomorphic encryption, and adversarial examples along with the brief working of an FHE-based ML framework that we use to demonstrate the possibility of model stealing attacks on secure MLaaS. Section 3 provides an overview of model-stealing attacks through adversarial examples in the context of secure MLaaS. Section 4 shows a demonstration of a model stealing attack on an FHE-only FCN model. Section 5 provides our evaluation of existing countermeasures against model stealing attacks using adversarial examples and their relevance in the context of secure MLaaS. Section 6 concludes our paper.

2 Preliminaries and Background

In this section, we provide the necessary background related to our work, starting with an introduction to ML model followed by an overview of FHE and an FHE-based ML framework. Finally, we introduce the concept of adversarial examples.

[3] The client controls both encryption and decryption using its secret key.

2.1 Machine Learning Model

A machine learning model can be viewed as a mapping function $f : \mathcal{X} \rightarrow \mathcal{Y}$, that maps the input feature space $\mathcal{X}$ to the output space $\mathcal{Y}$. An input $\mathbf{x} \in \mathcal{X}$ to the model is a d-dimensional vector. The actual inputs to the model come from a space $\mathcal{M}$, such as images or text, and the features are extracted by a function $\texttt{ef} : \mathcal{M} \rightarrow \mathcal{X}$. In classic MLaaS, this feature extraction can be done either at the client or server end since both can see the original input in the clear. However, in secure MLaaS, feature extraction takes place only on the client's end and the extracted features are encrypted before being sent to the server.

In this work, we primarily focus on classification setting, where the output of f is a single class label (termed as hard label in literature) that belongs to a set of classes $\mathbb{Z}_c$, i.e., $\mathcal{Y} \in \mathbb{Z}_c$. In certain applications, the labels are accompanied by *confidence scores* which signify the confidence of the model on each of the output labels. One application where the confidence scores come in handy is disease prediction, which takes the symptoms as inputs and outputs multiple disease labels along with the confidence scores of each. In this scenario, f outputs a vector of label-confidence pairs (termed as soft labels in literature), either for all labels (all possible diseases) or for top-k labels[4] (most probable diseases).

In literature, model training is done either in a supervised or unsupervised mode of learning. In supervised learning, the training data consists of input-output pairs $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$, upon which a training algorithm $\mathcal{T}$ is run. To ensure that the model is trained correctly, it is required that the training input $\mathbf{x}$ is labeled correctly. In practice, this label annotation is carried out by human experts and it consumes a significant cost. Coming back to the training process, each execution of $\mathcal{T}$ is a model f that consists of *parameters* such as weights and biases. In essence, f learns a mapping between the known input-output pairs, which it later applies to unseen data. The architecture of f can be divided into linear and non-linear layers. The output feature z_i of i^{th} linear layer is defined as $z_i = \mathbf{w}_i \cdot \mathbf{x}_i + b_i$, where $\mathbf{x}_i$ is the input feature vector and $\mathbf{w}_i$ and b_i are the corresponding weights and biases. The output of a linear layer is then fed into a non-linear layer that consists of activation functions such as ReLU [32], Hyperbolic Tangent (tanh), Sigmoid, and others, which decides the features that are to be propagated to the next levels.

2.2 Fully Homomorphic Encryption

Unlike traditional encryption schemes, FHE allows an entity to perform arbitrary computations directly over encrypted data while ensuring that the result also remains encrypted. This allows a client to encrypt its input before sending it to the model provider to run an inference service on it. It finally receives the inference result, still in encrypted form, and then decrypts it to obtain the result in clear. The security of existing FHE schemes [3,9] is based on the idea of noise, a random value that is added to the ciphertexts, which increases with each homomorphic encryption. Once it breaches a pre-defined threshold, the correctness of

[4] top-k labels are $k < c$ with highest confidence values sorted in descending order.

decryption cannot be guaranteed. To mitigate this, existing constructions either limit the depth of the circuit [3] or reduces the noise using bootstrapping [9].

A major problem with implementing ML models with FHE schemes is that they can only evaluate the linear layers, as they can be represented using polynomial operations which the FHE schemes natively support. On the other hand, it is difficult to evaluate non-linear layers since they require non-polynomial operations [15]. Earlier constructions chose to side-step this issue by replacing the non-linear layers with a polynomial approximation of the same [46]. However, this approach suffers from two major drawbacks. Firstly, they require retraining of the model with a new architecture where the non-linear layers are replaced by their polynomial approximations [46]. Secondly, this approximation causes a decrease in the accuracy of the model. Recent constructions based on bootstrappable FHE schemes do not suffer from these drawbacks as they can directly evaluate non-linear layers without the need for any approximations. Their construction is based on the idea of programmable bootstrapping (PBS), which allows the evaluation of an arbitrary function during the bootstrapping step itself [11]. This provides the benefit that no re-training is required and a previously trained model on clear data can be utilized to run inference on the encrypted data [12] without a significant drop in accuracy.

2.3 Concrete-ML

Concrete ML [30] is a privacy-preserving ML inference framework built upon TFHE [9] that operates over Boolean or Integer data. The framework provides a set of tools that allows the model owner to directly convert their trained model to work over encrypted data without the need for retraining. Moreover, the user is required to have minimal knowledge of the inner workings of this framework for them to use it. This allows the model owners to easily and swiftly transition to secure MLaaS without incurring significant overhead. However, this framework requires quantizing the input features, model weights, and any intermediate values to integers during inference since the underlying FHE scheme can operate only on integers. Moreover, non-linear layers are evaluated using programmable bootstrapping [11] which is the slowest step during the whole execution of the model. Thus, the total inference time is mostly dependent on the number of non-linear layers to be evaluated, which increases with the depth of the network and the number of nodes in each layer. Additionally, the accuracy of the model is dependent on the number of bits that are used to quantize the inputs, weights, and intermediate values.

2.4 Adversarial Examples

In literature, adversarial examples are defined as specially crafted inputs that cause misclassification by a victim model [17]. The process of generating these examples often starts with taking an original input to the model and then adding perturbations, either random or targeted, to the whole or part of the inputs. Adversarial examples are generated to either force the victim model to output a specific label (targeted attacks) or any label apart from the original one (untargeted attacks). However, they are crafted in a manner that they remain imperceptible to humans lest they get detected.

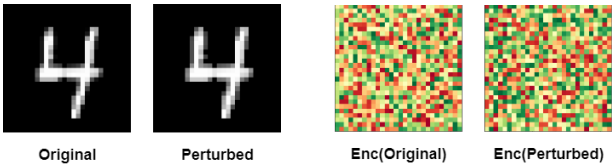
In adversarial attacks, the attacker is assumed to have either white-box access or black-box access to the victim model [27]. In white-box attacks, the adversary has knowledge about the victim model, including the model architecture, parameters and hyperparameters, and the training data. In black-box attacks, the adversary has no knowledge about the victim model and it can only access the model through prediction APIs. There is also a third category of attacks called the gray-box attacks where the adversary knows some information about the victim model, such as model architecture [42].

3 Model Stealing Through Adversarial Examples

The accuracy of a model is dependent on the quality of the dataset upon which it is trained. However, having access to the labeled dataset is itself a challenge due to two reasons. First, it involves the collection of unlabeled data, which often contains private information and thus is required to be kept confidential. Secondly, the collected data need to be properly labeled, which is a cumbersome process and requires manual intervention. However, the adversary (which is the client in this case) can be in possession of some of its own properly labeled dataset that it can use to verify the quality of service by the server. Such scenario can be considered as analogous to Known Answer Test (KAT) used in cryptographic schemes to verify the correctness of the scheme.[5]

It must be noted that such labeled datasets are usually small in size and do not contain enough samples to properly train a model. Moreover, the motivation for model stealing also includes the fact that training sophisticated models on large datasets requires lots of resources in terms of hardware, time and power. In literature, it has been shown that an adversary can utilize these samples to conduct model-stealing attacks in classic MLaaS settings [22,50]. The adversary first obtains the prediction values on the original samples by querying the target model. Next, it generates *synthetic data points* using existing adversarial example generation methods [22] and obtains their corresponding prediction values by querying the target model on these synthetic data points. Once enough samples are obtained, the adversary uses them as its training set to train a local model. However, this approach poses two major challenges to the adversary. First, the adversary is required to pay the victim model for each query made. Thus, a constraint is imposed on the adversary to carry out the attack at the minimum possible cost. There exist works like [41, 49, 50] that propose methods to carry out the attack with minimum cost. However, this constraint is not necessarily required in practice. The reason is that the incurred cost can be amortized after the model has been stolen since the adversary will no longer need to pay the victim model to avail of its service [41]. Moreover, it can host the stolen model as its own pay-per-use service and thus recover the cost [43]. Secondly, and more importantly, it is assumed in the literature that an adversary that is trying to steal the model often queries the model on similar inputs over a

[5] Cryptographic algorithms are often accompanied by a set of test vectors that aid in verifying the correctness of the implementation of the algorithm by ensuring whether the "random looking" outputs of these algorithms are indeed correct or not.



(a) Classic MLaaS (b) Secure MLaaS

Fig. 1: View of the original image and perturbed image from the server's perspective in (a) Classic MLaaS, where the server can see the images in clear, and (b) Secure MlaaS settings, where the server only sees the encrypted images.

period of time. For example, in the case of image processing tasks, the adversary often queries the model with similar images with a small difference in added perturbation. In classic MLaaS, such similar inputs can often be detected by the victim model. However, such challenges do not apply in secure MLaaS scenarios. Since the server hosting the model receives the inputs in encrypted form, it cannot observe the similarities between the inputs. We highlight this difference in Fig. 1, where the left and right pairs of images show the view of these images from the perspective of the server in classic and secure MLaaS settings, respectively. Due to the inability of the model to differentiate between an original and a perturbed image, or even between two original images, the adversary is free to query the model on any image of its choice, including similar ones. This also holds true for other input domains.

4 Model-Stealing Attack on Secure MLaaS Application

In this section, we elaborate on our threat model and demonstrate model stealing attack on FHE-only secure MLaaS applications.

4.1 Threat model

We assume that the adversary has gray-box access to the victim model where it only knows the architecture of the model [42]. The adversary has access to a small set of its own labeled samples. Such labeled datasets are available to the adversary (which is the client for a secure MLaaS setting) in practice either as a proprietary dataset or, in certain cases, the model providers often supply the clients with a few test samples with correct labels, that may be part of the validation dataset, which they can use to first test the accuracy of the model for free before availing the service. Such assumptions are in line with previous works in literature [16] in the context of malware detectors and objectionable content detectors. In the first case, the adversary possesses known malware samples that it wants to perturb in order to get them misclassified as a benign sample. In the second case, the adversary is in possession of known objectionable content that it wants to perturb in order to get them misclassified as normal images so that they can be uploaded on some website without getting censored. In both these examples, the adversary already knows the correct label of the samples it possesses, which it can use to steal the model. We further assume that the model only outputs the predicted class label and not the associated confidence

Algorithm 1 Model Stealing using Adversarial Samples

```
X := matrix of original samples
y := vector of labels corresponding to original samples
ϵ := perturbation to be added
bgcol := colour corresponding to background pixel
f_v := victim model
f_s := stolen model
pert_type := random or targeted
function getPerturbedSample(x, ϵ, bgcol, pert_type)
    x' := ∅
    for x in x do
        if pert_type == "random" then                    ▷ All pixels are perturbed
            x ← x + ϵ
        end if
        if pert_type == "targeted" then                  ▷ Specific pixels are perturbed
            if x != bgcol then                           ▷ Pixel is not part of background
                x ← x + ϵ
            end if
        end if
        x' ← x'||x
    end for
    return x'
end function
function runAttack(X, y, ϵ, bgcol, f_v, pert_type)
    X' := ∅, y' := ∅
    for x in X do
        x' ← getPerturbedSample(x, ϵ, bgcol, pert_type)
        y' ← f_v(x')
        X' ← X'||x'
        y' ← y'||y'
    end for
    f_s ← Train(X||X', y||y')                            ▷ Train is some training algorithm
    return f_s
end function
```

values. We note that this assumption makes it difficult for the adversary to clone a model as providing extra information to the adversary, apart from the class labels, makes it easier to steal the model with fewer queries, as shown in previous works in the context of classic MLaaS [41].

4.2 Attack Methodology

We now demonstrate a model-stealing attack on an existing secure MLaaS application built using the Concrete-ML framework [30]. However, we would like to highlight that our attack is not dependent on the intricacies of the targeted framework, which is chosen due to its public availability, and it can be carried out on any FHE-based ML framework. Coming back to our attack, the applica-

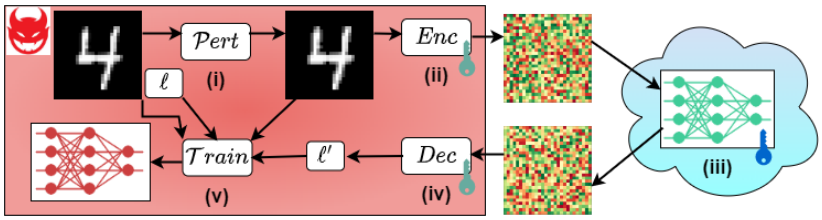


Fig. 2: Our end-to-end attack process. The adversary (i) perturbs a clean image using some algorithm $\mathcal{P}ert$, (ii) encrypts the perturbed image, (iii) sends it to the server and receives the encrypted result, (iv) decrypts it to obtain predicted label in clear, (v) once enough samples are obtained, the original and adversarial samples along with corresponding labels are fed into some training algorithm $\mathcal{T}rain$ to obtain the stolen model.

tion[6] we target is based on handwritten digit identification, which is trained on the MNIST [14] dataset. The model outputs a hard label that corresponds to the digit in the input image. The application uses a 2 layer FCN that uses Sigmoid as its activation function. The inputs and weights are quantized[7] using 4 bits while the size of the intermediate accumulator is set to 15 bits, one short of the maximal value of 16 bits currently supported by Concrete-ML. The adversary is aware of the number of bits used to quantize the inputs since quantization takes place on the client's side and the quantized inputs are then encrypted. Moreover, the adversary may assume the number of bits used by the server to quantize its weights to be the same as that was used to quantize the inputs. The assumption is valid since each linear layer computation in the model takes place over a pair consisting of a client input and a model weight. Finally, the adversary can safely assume that the victim model uses the highest possible size of the accumulator. The reason is that this accumulator is used by the server to store the intermediate results of a linear layer before a non-linear operation can be performed. Moreover, the server does not know the exact number of bits that will be required to represent the underlying plaintext value of the corresponding intermediate value, since the server cannot see it in the clear. Thus, to prevent the chance of any overflows, it is better to use the highest possible bit-length that is currently supported by the framework. We now explain the methodology to generate the adversarial samples and then show the results in terms of the number of queries required and the accuracy of the stolen model. The end-to-end process of our attack is shown in Fig. 2 and explained in Algorithm 1.

We assume wlog that the first 1000 images of the labeled dataset are available to the adversary as part of the validation dataset. The samples are first pre-processed before adding any perturbations. This is done by first scaling down each pixel value of the images by the maximum pixel value, which is originally

[6] It can be found at https://github.com/zama-ai/concrete-ml/blob/release/1.1.x/docs/advanced_examples/FullyConnectedNeuralNetworkOnMNIST.ipynb

[7] Quantization is required to convert the real-valued inputs and weights to integer values since FHE schemes currently operate only on integer data.

255. The mean value is then subtracted from each pixel value, and the obtained value is further scaled down by the standard deviation. The final post-processed values are obtained by rounding the scaled-down pixels to 4 places of decimal. These pre-processing steps are the same as defined in the original application. Once we obtain the set of post-processed images $\mathbf{X}$ along with their original labels $\mathbf{y}$, we proceed with perturbing these images to generate adversarial samples. This is done by adding perturbation values ϵ ranging from 0.1 to 1.0, in increments of 0.1, to each of the post-processed images $\mathbf{x}$ to obtain perturbed version $\mathbf{x}'$.

We adopt two different strategies to generate the adversarial samples. In the first strategy, we perform `random` perturbations by adding the perturbation values in all the pixels (including the ones defining the background) as shown in lines $11 - 13$ of Algorithm 1. In the second strategy, we perform `targeted` perturbation by adding the perturbation values in only those pixels that form part of the digit and are not part of the background as shown in lines $14 - 18$. Contrary to [16] where the authors perturb the background pixels, we target the pixels representing the digits. Since the background of the MNIST images is set to black, all the corresponding pixels in the post-processed image consist of the same value, which is denoted as *bgcol* (ref. Algorithm 1). We thus identify the pixels that form the part of the digit to be those whose value is not equal to *bgcol*. Once the adversarial image set $\mathbf{X}'$ is obtained, we query the victim model f_v on them to obtain their predicted class labels $\mathbf{y}'$.

For our experiment, we added a perturbation value of 0.1 to each of our 1000 (without loss of generality) samples and queried the victim model f_v to obtain the corresponding labels. We repeated this process to obtain 10 batches of adversarial samples, each of size 1000. The first batch was obtained by adding a perturbation value of 0.1, and for the rest of the batches, the perturbation value was consecutively incremented by 0.1. Finally, we trained 10 different models using the same training algorithm $\mathcal{T}rain$ (ref. Algorithm 1). The first model was trained on the augmented set of size 2000 that consists of original 1000 samples and the first batch of 1000 adversarial samples, which was obtained by adding a perturbation value of 0.1. For the second model, we further augmented the second batch of 1000 adversarial samples, which was obtained by adding a perturbation value of 0.2, to the previously augmented dataset which increased the size of the overall dataset to 3000. This process was repeated to obtain all 10 models, after which the model with the highest accuracy was chosen as the stolen model f_s. We updated the architecture of the stolen model by increasing the number of fully connected layers and the number of bits for quantizing inputs and weights to 4 and 6 from the original 2 and 4, respectively. We chose these values as they provided good accuracy of the stolen model without increasing the depth of the network by much.

4.3 Experimental Results

Fig. 3 shows the accuracy of the stolen model with random (shown in green) and targeted (shown in red) perturbations, respectively. For the first case, we obtain the maximum accuracy of 85% when the added perturbation ranges from 0.1 to 0.3 (both inclusive), thus requiring 3000 queries in total to the victim model.

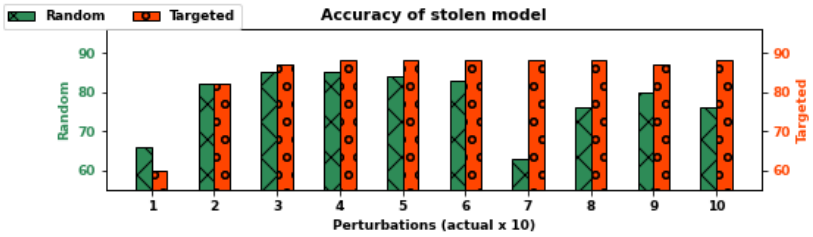


Fig. 3: Plot of the accuracy of the stolen model in random (green) vs. targeted (red) perturbations. The added perturbations are cumulated when going from left to right along the x-axis.

For the second case, we obtain the maximum accuracy of 88% when the added perturbation ranges from 0.1 to 0.4 (both inclusive), thus requiring 4000 queries in total to the victim model. We would like to highlight that while we require an additional 1000 queries in the second case, these extra queries resulted in an increase of 3% in the accuracy of the stolen model. It is worth mentioning that the client can make as many queries as it wants after making the requisite payment, as is the case for any pay-per-use service. Moreover, since all the input queries are encrypted, each query appears as distinct to the server. To calculate the accuracy of the cloned model, we perform inference (testing) on the cloned model using the training set of the original model. One must note that the attack does not require the availability of original training set to the adversary. It is only used to calculate the accuracy of the stolen model. To compare the loss of accuracy, we obtained the accuracy score of the victim model over the testing data, which turned out to be 96%. However, we would like to highlight that the victim model was trained on 50000 clean samples. On the other hand, we obtained a stolen model with far fewer training samples, totaling 5000, of which 1000 are clean samples and the rest 4000 are adversarially generated, with a loss of 7% in accuracy over the original dataset.

4.4 Attack against Robust ML models

To increase the robustness of the model against adversarial attacks, i.e., to prevent misclassification of adversarial samples by the victim model, a number of works [17, 29, 33, 39] proposed to augment adversarial examples in the training dataset. The adversarial samples are generated by perturbing a clean sample using known adversarial example generation techniques, which are then classified with the labels of the original sample. This ensures that the model learns the relationship between an adversarial sample and its original label so that any future query with an adversarial sample is responded with the correct label and not the wrong one. However, the impact of this augmentation on the accuracy of the stolen model has not been studied in the literature. To evaluate the same, we performed two more experiments. In the first experiment, we first train the victim model using the original training samples. We then use the first 10000 training points to generate local adversarial samples, assuming that the victim model knows the sample generation technique of the adversary. Similar to the

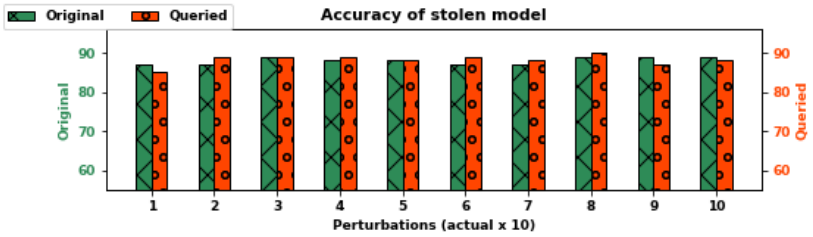


Fig. 4: Plot of the accuracy of the stolen model when the victim model incorporates adversarial examples with queried labels (green) vs. when it incorporates adversarial examples with original labels (red). The added perturbations are cumulated when going from left to right along the x-axis.

previous attack method, we generate the adversarial samples by adding perturbations into the pixels corresponding to the digit in increments of 0.1 in the range 0.1 to 1.0. Thereafter, these adversarial samples are labeled with that of their corresponding original samples. Next, we train a model over the original training data along with these adversarial samples. More specifically, we train 10 different models, with the first model trained on original training data plus the adversarial data obtained by adding the perturbation value of 0.1, the second one on original training data plus the adversarial data obtained by adding the perturbation values of 0.1 and 0.2, and so on. Each model was tested over the original testing dataset. We obtained the highest accuracy of 97% after incorporating the adversarial samples obtained by adding the perturbation values from 0.1 to 0.9 (inclusive).

Considering the newly generated model as the target model, we repeat the model stealing attack process as per Algorithm 1. Additionally, we perform the second experiment in a similar manner, with the only difference being that we first train the victim model using only the original training data upon which we query the adversarial samples to obtain their labels, and then we retrain the victim model on both the original as well as the adversarial samples. The motivation behind the second experiment is to study the impact on the accuracy of the stolen model when the victim model contains the adversarial samples along with their misclassified labels. Fig. 4 shows the accuracy of the stolen model when the victim model incorporated adversarial examples with the original labels (shown in green) and when the victim model incorporated adversarial examples with labels obtained by querying the original model (shown in red). For the first case, we obtained the highest accuracy of 89% when using samples with perturbation values ranging from 0.1 to 0.3, thus requiring a total of 3000 queries to the victim model. For the second case, we obtained the highest accuracy of 90% when using samples with perturbation values ranging from 0.1 to 0.8, thus requiring a total of 8000 queries to the victim model.

From Fig. 3 and Fig. 4, we can observe that incorporating adversarial samples during training of the victim model does not affect the accuracy of the stolen model. Moreover, we observe that augmenting the adversarial samples in the victim model leads to an increase in the accuracy of the stolen model by 1%

when they were labeled with that of their corresponding clean samples, and 2% when the adversarial samples are labeled according to the prediction results of the original model, albeit requiring an increase in the number of queries to the victim model. Even with a lower number of queries, we observed that the accuracy of the stolen model is similar irrespective of whether the adversarial samples were augmented in the stolen model or not. This shows that augmenting adversarial samples during the training of the victim model does not hamper the accuracy of the stolen model. On the contrary, it may even lead to an increase in the accuracy of the stolen model.

5 Evaluation of Existing Defense Methods

In this section, we evaluate existing countermeasures against model-stealing attacks using adversarial examples in the context of secure MLaaS. We first discuss the countermeasures that are deployed to detect whether an adversary is querying the model on adversarial samples. We then discuss the methods that are used to detect whether a suspect model is a stolen version of some victim model or not. We further categorize the works based on their underlying primary assumption.

5.1 Defenses based on query distributions

This class of defenses is based on the assumption that the adversary queries the model using only the adversarial samples to reduce its number of queries [49]. Moreover, it is assumed that the inputs look almost the same in case they are images [8]. Finally, it is assumed that the distribution from which these adversarial samples are drawn is different from the distribution from which the actual inputs are drawn [37]. Thus these defenses try to measure the statistical difference between the adversarial and its corresponding natural inputs or between two consecutive inputs to check whether they are too far or too close, respectively.

Authors in [22] proposed DefenseNet, a network that analyzes the distribution of queries to victim models by analyzing the difference between the distributions of features between the adversarial and benign images. The features were extracted from each hidden layer's output. However, this defense does not work on secure ML applications. The reason is that FHE ensures that the results of some homomorphic computation over encrypted data also remain encrypted. This implies that the extracted features are themselves encrypted since they are the results of homomorphic operations over encrypted inputs. Moreover, any distance computation over them results in encrypted statistics that can only be decrypted by the owner of the secret key, which is the adversary itself.

Similarly, [35] proposed a method that is based on identifying a sparse set of features that defines the output prediction result and altering which can easily alter the prediction value. They do so by searching for the closest decision boundaries of a point in the feature space that is most important for the prediction result and then using these points to train a surrogate model that approximates these boundaries. However, this method will not work in secure MLaaS for the same reason that running the search on encrypted points will result in encrypted boundary values that the server cannot observe directly. Moreover, the current

frameworks for secure MLaaS only support training over cleartext data and not encrypted ones, which ensures that the server cannot train the surrogate model over the encrypted decision boundary values.

Another work [29] proposed training a set of models for individual layers of the network. The models try to predict how a set of activated neurons in one layer causes a change in the set of activated neurons in the next layer, where these changes vary when the input features are adversarial and when they are clean. However, this requires training on encrypted values which the current secure MLaaS frameworks do not support. Moreover, the decision as to whether the input sample is adversarial or clean is taken based on the prediction results of these models, which themselves are encrypted and not observable by the server.

Based on the assumption that black-box model-stealing attacks make a sequence of similar queries, where each input differs from the next one by a small distance, [8] proposed a similarity-detector neural network model that can identify these query patterns. Their idea is to temporarily record and store each query in a buffer, and then for each new query, compute the distance between this query and the previously stored ones. They flag a query if the distance falls below a pre-defined threshold. However, in the context of secure MLaaS, this computation of distance needs to be done homomorphically over encrypted data with the computed distance itself being encrypted and not observable by the server without access to the secret key of the adversary.

Recently, [33] proposed a method that is based on the concept of denoising, which aims to reduce the noise in the input sample. Once done, they compute the difference between the original adversarial sample and its approximation. It identifies the input sample to be adversarial if this difference breaches a certain threshold. However, in the secure MLaaS setting, this difference will be computed homomorphically and thus will be encrypted, requiring access to the secret key for decrypting and observing the value in the clear.

Another recent work [1] proposed training two different models, one on the original images and one on the transformed images. They then query both the models on the received input to obtain two prediction results. Finally, they compare the two results and denote the received input to be adversarial if they are different, otherwise, they consider it to be benign. The intuition behind their method is that the decision boundary of both models will be different since they are trained on different datasets, which will make it difficult for the adversary to come up with a single adversarial sample that gets misclassified to the same label by both models. However, this method will not work in the secure MLaaS setting since the outputs of both models will be encrypted, and thus their similarity or dissimilarity will not be observable by the server.

5.2 Defenses based on perturbing probability values

These defenses are based on the fact that providing the adversary with additional information in the form of confidence values can lead to model stealing attacks with a reduced number of queries [25]. Thus these defenses aim to perturb the confidence values, without changing the accompanying class labels, in order to force the adversary to only use the class labels.

Authors in [23] proposed a heuristic-based technique that builds upon the technique of [25] but used a different function to calculate the perturbation to be added since the original technique added a high perturbation that affected the transparency of the model. Once obtained, they injected the noise in only the probabilities with low values since they symbolize the low confidence of the model on the accompanying class label. However, this method will not work in the secure MLaaS setting since the server cannot see which probability values are high and which are low, since the values will be encrypted. [19] proposed two methods of altering the final confidence values without affecting the class labels. They do so by either changing the temperature coefficient of the final softmax layer or adding noise values drawn from a normal distribution with some variance. They highlight that the strength of the defense increases with an increase in the variance. [44] proposed adding perturbation to the confidence score vector of the final layer instead of the final probability vector. To generate these perturbation values, they train a local model on adversarial samples and then use its gradient to find these values. Their goal is to maximize the distance between the original and perturbed confidence values, which ensures that the highest probability value still remains highest.

However, the aforementioned methods are reserved for only those models that output confidence values along with the class labels. Moreover, we have already highlighted in Section 3 that increasing the cost of the attack, which forms the foundation of these defenses by forcing the adversary to use only class labels, is not a major constraint since the adversary can recover the incurred cost by hosting the stolen model and offer it as a service. Finally, while perturbing the output is allowed in the classic MLaaS setting, it is not allowed in the secure MLaaS setting. The reason is that this setting considers the server to be *semi-honest* which is not allowed to tamper with the final output, or for that matter any data involved in the computation. If allowed to do so, a malicious server can utilize it to perform interactive key-recovery attacks [5, 10] which will cause a breach in confidentiality of the data of an honest client.

5.3 Defenses based on identifying stolen models

These are not defenses per se, as these methods are invoked after the model has already been stolen and are used to confirm whether a suspect model is a stolen one or a homologous one [34].[8] These defenses make use of either watermarking techniques or decision boundary identifications. In this work, we only focus on methods that use the decision boundary identification techniques since the watermarks can be easily removed from the stolen model by an adversary [18]. Before proceeding, we would like to highlight that the decision boundary identification-based methods assume that DNNs can be characterized by their unique classification boundaries. This suggests that a stolen model will share its decision boundaries with the victim model, which will produce the same class labels when queried upon with the same adversarial samples [43]. The

[8] A homologous model is one which is trained on the same training dataset as the victim model but differs in terms of decision boundaries.

methods we evaluate here only differ in the way they generate the adversarial samples, which we provide here for brevity.

Authors in [43] generated the target samples by first randomly selecting n points that are closest to the decision boundary of a chosen class label t. They then add minimal perturbations to these random values such that querying their model on these perturbed samples outputs t as the class label. On the contrary, [6] starts with a clean sample x and then adds a large, random perturbation to it such that querying its model on this perturbed sample x'' causes a misclassification and outputs a different label. They then perform a binary search between x and x'' to obtain the target sample x' on the decision boundary. [34] first generates a universal adversarial perturbation [31] (UAP) for its own model. They then perform a k-means clustering on the representation vectors of its training data to cluster them into n clusters. Finally, they pick one data point from each cluster and add the UAP to it to obtain the adversarial sample x'. [47] first decides a specific class t with a fixed confidence value c, and then uses a modified version of C & W method [4] to obtain the adversarial sample x' that classifies to t.

In all the aforementioned methods, the pair (x', t), where x' is the adversarial sample and t is the original label, acts as the model fingerprint. To verify whether a suspect model is stolen or not, it is queried on multiple such pairs to obtain a vector of labels t', after which the similarity between the vectors t and t' is computed. The suspect model is deemed to be stolen if the similarity score turns out to be large, otherwise, it is deemed to be a homologous model [34]. The reason the aforementioned methods work in the secure MLaaS setting is that the roles of the adversary and the victim model are reversed, as now the stolen model acts as the victim model while the owner of the victim model acts as the adversary. Moreover, the stolen model is queried over encrypted data which hides the adversarial samples, since the detection techniques deployed by the stolen model will not work for reasons already highlighted in Section 5.1 and 5.2.

6 Conclusion

This paper highlights that model stealing is possible in even secure MLaaS settings. We highlight that existing techniques for model stealing using adversarial samples that work in the classic MLaaS settings also apply to secure MLaaS settings. Moreover, we show that the existing constraints applicable to model stealing in classic MLaaS can be relaxed in the context of secure MLaaS as the encryption of inputs allows us to do so. Finally, our extensive review of existing countermeasures for classic MLaaS shows that they are not sufficient to prevent model stealing attacks in secure MLaaS. This warrants further study into possible countermeasures to detect these attacks in secure MLaaS-based applications since these applications are almost gearing up for production deployment.

References

1. Alam, M., Datta, S., Mukhopadhyay, D., Mondal, A., Chakrabarti, P.P.: Resisting adversarial attacks in deep neural networks using diverse decision boundaries.

arXiv preprint arXiv:2208.08697 (2022), https://arxiv.org/abs/2208.08697
2. Albawi, S., Mohammed, T.A., Al-Zawi, S.: Understanding of a convolutional neural network. In: 2017 international conference on engineering and technology (ICET). pp. 1–6. Ieee (2017)
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. vol. 6, pp. 1–36. ACM New York, NY, USA (2014)
4. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 ieee symposium on security and privacy (sp). pp. 39–57. Ieee (2017)
5. Chaturvedi, B., Chakraborty, A., Chatterjee, A., Mukhopadhyay, D.: vr^{2}fhe- securing fhe from reaction-based key recovery attacks. Cryptology ePrint Archive, Paper 2023/561 (2023), https://eprint.iacr.org/2023/561
6. Chen, J., Jordan, M.I.: Boundary attack++: Query-efficient decision-based adversarial attack. arXiv preprint arXiv:1904.02144 **2**(7) (2019), http://arxiv.org/abs/1904.02144
7. Chen, S., Fan, J.: Seek: model extraction attack against hybrid secure inference protocols (2022), https://eprint.iacr.org/2022/1200
8. Chen, S., Carlini, N., Wagner, D.: Stateful detection of black-box adversarial attacks. In: Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence. pp. 30–39 (2020)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
10. Chillotti, I., Gama, N., Goubin, L.: Attacking fhe-based applications by software fault injections. Cryptology ePrint Archive, Paper 2016/1164 (2016), https://eprint.iacr.org/2016/1164
11. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5. pp. 1–19. Springer (2021)
12. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5. pp. 1–19. Springer (2021)
13. Correia-Silva, J.R., Berriel, R.F., Badue, C., De Souza, A.F., Oliveira-Santos, T.: Copycat cnn: Are random non-labeled data enough to steal knowledge from black-box models? Pattern Recognition **113**, 107830 (2021)
14. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
15. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International conference on machine learning. pp. 201–210. PMLR (2016)
16. Gilmer, J., Adams, R.P., Goodfellow, I., Andersen, D., Dahl, G.E.: Motivating the rules of the game for adversarial example research (2018)
17. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014), http://arxiv.org/abs/1412.6572
18. Guo, S., Zhang, T., Qiu, H., Zeng, Y., Xiang, T., Liu, Y.: Fine-tuning is not enough: A simple yet effective watermark removal attack for dnn models. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21. pp. 3635–3641. International Joint Conferences on Artificial Intelligence Organization (2021), https://doi.org/10.24963/ijcai.2021/500

19. He, X., Lyu, L., Sun, L., Xu, Q.: Model extraction and adversarial transferability, your bert is vulnerable! In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 2006–2012 (2021)
20. Huang, J., Chai, J., Cho, S.: Deep learning in finance and banking: A literature review and classification. Frontiers of Business Research in China **14**, 1–24 (2020)
21. Huang, Z., Lu, W.j., Hong, C., Ding, J.: Cheetah: Lean and fast secure {two-party} deep neural network inference. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 809–826 (2022)
22. Juuti, M., Szyller, S., Marchal, S., Asokan, N.: Prada: protecting against dnn model stealing attacks. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 512–527. IEEE (2019)
23. Khaled, K., Dhaouadi, M., de Magalhães, F.G., Nicolescu, G.: Efficient defense against model stealing attacks on convolutional neural networks. arXiv preprint arXiv:2309.01838 (2023), https://arxiv.org/abs/2309.01838
24. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: Secure tensorflow inference. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 336–353. IEEE (2020)
25. Lee, T., Edwards, B., Molloy, I., Su, D.: Defending against neural network model stealing attacks using deceptive perturbations. In: 2019 IEEE Security and Privacy Workshops (SPW). pp. 43–49. IEEE (2019)
26. Lehmkuhl, R., Mishra, P., Srinivasan, A., Popa, R.A.: Muse: Secure inference resilient to malicious clients. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2201–2218 (2021)
27. Li, Y., Guo, Y., Xie, Y., Wang, Q.: A survey of defense methods against adversarial examples. In: 2022 8th International Conference on Big Data and Information Analytics (BigDIA). pp. 453–460. IEEE (2022)
28. Li, Z., Zhang, Y.: Membership leakage in label-only exposures. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 880–895 (2021)
29. Ma, S., Liu, Y., Tao, G., Lee, W.C., Zhang, X.: Nic: Detecting adversarial samples with neural network invariant checking. In: 26th Annual Network And Distributed System Security Symposium (NDSS 2019). Internet Soc (2019)
30. Meyre, A., Chevallier-Mames, B., Frery, J., Stoian, A., Bredehoft, R., Montero, L., Kherfallah, C.: Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists (2022), https://github.com/zama-ai/concrete-ml
31. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1765–1773 (2017)
32. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 807–814 (2010)
33. Pauling, C., Gimson, M., Qaid, M., Kida, A., Halak, B.: A tutorial on adversarial learning attacks and countermeasures (2022)
34. Peng, Z., Li, S., Chen, G., Zhang, C., Zhu, H., Xue, M.: Fingerprinting deep neural networks globally via universal adversarial perturbations. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 13420–13429. IEEE Computer Society (2022)

35. Renard, X., Laugel, T., Lesot, M.J., Marsala, C., Detyniecki, M.: In: Workshop on Recent Advances in Adversarial Learning (Nemesis) of the European Conference on Machine Learning and Principles of Practice of Knowledge Discovery in Databases (ECML-PKDD) (2018)
36. Shailaja, K., Seetharamulu, B., Jabbar, M.: Machine learning in healthcare: A review. In: 2018 Second international conference on electronics, communication and aerospace technology (ICECA). pp. 910–914. IEEE (2018)
37. Song, Y., Kim, T., Nowozin, S., Ermon, S., Kushman, N.: Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. arXiv preprint arXiv:1710.10766 (2017), http://arxiv.org/abs/1710.10766
38. Stoian, A., Frery, J., Bredehoft, R., Montero, L., Kherfallah, C., Chevallier-Mames, B.: Deep neural networks for encrypted inference with tfhe. In: International Symposium on Cyber Security, Cryptology, and Machine Learning. pp. 493–500. Springer (2023)
39. Stutz, D., Hein, M., Schiele, B.: Disentangling adversarial robustness and generalization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6976–6987 (2019)
40. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
41. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction {APIs}. In: 25th USENIX security symposium (USENIX Security 16). pp. 601–618 (2016)
42. Vivek, B., Mopuri, K.R., Babu, R.V.: Gray-box adversarial training. In: Proceedings of the European conference on computer vision (ECCV). pp. 203–218 (2018)
43. Wang, S., Chang, C.H.: Fingerprinting deep neural networks-a deepfool approach. In: 2021 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5. IEEE (2021)
44. Wen, J., Yiu, S.M., Hui, L.C.: Defending against model inversion attack by adversarial examples. In: 2021 IEEE International Conference on Cyber Security and Resilience (CSR). pp. 551–556. IEEE (2021)
45. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016), https://arxiv.org/abs/1609.08144
46. Xie, P., Bilenko, M., Finley, T., Gilad-Bachrach, R., Lauter, K., Naehrig, M.: Crypto-nets: Neural networks over encrypted data. arXiv preprint arXiv:1412.6181 (2014), http://arxiv.org/abs/1412.6181
47. Xue, M., Sun, S., He, C., Zhang, Y., Wang, J., Liu, W.: Activeguard: An active dnn ip protection technique via adversarial examples. arXiv preprint arXiv:2103.01527 (2021), https://arxiv.org/abs/2103.01527
48. Yi, H., Shiyu, S., Xiusheng, D., Zhigang, C.: A study on deep neural networks framework. In: 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). pp. 1519–1522. IEEE (2016)
49. Yu, H., Yang, K., Zhang, T., Tsai, Y.Y., Ho, T.Y., Jin, Y.: Cloudleak: Large-scale deep learning models stealing through adversarial examples. In: NDSS (2020)
50. Yuan, X., Ding, L., Zhang, L., Li, X., Wu, D.O.: Es attack: Model stealing against deep neural networks without data hurdles. IEEE Transactions on Emerging Topics in Computational Intelligence **6**(5), 1258–1270 (2022)

51. Zhou, M., Wu, J., Liu, Y., Liu, S., Zhu, C.: Dast: Data-free substitute training for adversarial attacks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 234–243 (2020)
52. Zhu, Y., Cheng, Y., Zhou, H., Lu, Y.: Hermes attack: Steal {DNN} models with lossless inference accuracy. In: 30th USENIX Security Symposium (USENIX Security 21) (2021)