Chinese Journal of Electronics
vol. 33, no. 1, pp. 90–100, January 2024
https://doi.org/10.23919/cje.2022.00.377

RESEARCH ARTICLE

SwiftTheft: A Time-Efficient Model Extraction Attack Framework Against Cloud-Based Deep Neural Networks

Wenbin YANG[1], Xueluan GONG[2], Yanjiao CHEN[3], Qian WANG[1], and Jianshuo DONG[1]

1. *School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China*
2. *School of Computer Science, Wuhan University, Wuhan 430072, China*
3. *College of Electrical Engineering, Zhejiang University, Hangzhou 310058, China*

Corresponding author: Qian WANG, Email: qianwang@whu.edu.cn
Manuscript Received November 5, 2022; Accepted March 13, 2023
Copyright © 2024 Chinese Institute of Electronics

Abstract — With the rise of artificial intelligence and cloud computing, machine-learning-as-a-service platforms, such as Google, Amazon, and IBM, have emerged to provide sophisticated tasks for cloud applications. These proprietary models are vulnerable to model extraction attacks due to their commercial value. In this paper, we propose a time-efficient model extraction attack framework called SwiftTheft that aims to steal the functionality of cloud-based deep neural network models. We distinguish SwiftTheft from the existing works with a novel distribution estimation algorithm and reference model settings, finding the most informative query samples without querying the victim model. The selected query samples can be applied to various cloud models with a one-time selection. We evaluate our proposed method through extensive experiments on three victim models and six datasets, with up to 16 models for each dataset. Compared to the existing attacks, SwiftTheft increases agreement (i.e., similarity) by 8% while consuming 98% less selecting time.
Keywords — Artificial intelligence security, Model extraction attacks, Deep neural networks.
Citation — Wenbin YANG, Xueluan GONG, Yanjiao CHEN, *et al.*, "SwiftTheft: A Time-Efficient Model Extraction Attack Framework Against Cloud-Based Deep Neural Networks," *Chinese Journal of Electronics*, vol. 33, no. 1, pp. 90–100, 2024. doi: 10.23919/cje.2022.00.377.

I. Introduction

Deep learning has recently made a significant breakthrough in various applications, including license plate reading, disease diagnosing, and even sophisticated autopilot. However, deep learning models require an enormous amount of training samples and computing resources to reach high prediction performance. To alleviate the burdensome data collection and training process, various cloud-service providers like IBM, Amazon, Microsoft, and Google host sophisticated DNN models on the cloud to provide machine-learning-as-a-service (MLaaS). Ordinary clients utilize MLaaS for retrieving expected predictions by submitting queries through the API interface. Any technical details of the training data, model architecture, and model hyperparameters are inaccessible to the users. Therefore, such a cloud-based model is treated as a black box for end-user applications.

However, various studies have reported that high-value black-box models are vulnerable to model extraction attacks [1], [2]. The motivations of model extraction attacks are mainly two-fold. On the one hand, the attacker can obtain commercial value by reselling the cloud-based model. On the other hand, the substitute model can be utilized as a springboard for further attacks, e.g., adversarial example attacks [3], backdoor attacks [4].

To recap, Tramèr *et al.* [5] proposed the first model extraction attack that is effective for various simple machine learning models, e.g., logistics regression, support vector machine (SVM), decision tree, and shallow neural network (NN). However, such a method is not applicable for deep neural networks. Wang and Gong [6] proposed the first hyperparameter extraction attack against the black-box models. Duddu *et al.* [7] proposed a side-channel attack that aims to steal the model structure.

Associate Editor: Prof. Yidong LI, Beijing Jiaotong University.

Papernot *et al.* [3] proposed a Jacobian-based method to augment query samples. CloudLeaks [8] exploited adversarial examples to extract decision boundaries with the help of a subset of original training data. However, both Papernot *et al.*'s strategy and CloudLeaks require a subset of training data to achieve respectable attack performance, which breaks the premise of the black-box setting. Copycat CNN [9] uses non-problem domain natural samples to conduct the model extraction attacks. However, Copycat CNN will incur a high query budget since it only randomly selects samples from the public dataset as the query samples. More recently, Knockoff nets [2] and ActiveThief [1] utilize active learning and reinforcement to select a proper subset from public datasets to reduce the query budgets. However, both ActiveThief and Knockoff nets use an iterative selection approach, which involves burdensome model training in each iteration.

In this work, we design SwiftTheft, a model extraction method against cloud-based deep neural networks (DNNs). SwiftTheft employs reference models, which are neural networks that take in an image and output its corresponding representations, to efficiently select representative samples from a large dataset in a single step without iterative training. These samples are used to query the black-box model and train the substitute model on the queried results, enabling SwiftTheft to be applied to various cloud models and significantly improving efficiency compared to existing methods. Specifically, SwiftTheft trains multiple reference models on public data using non-overlapping subsets of the data and different parameter initializations (referred to as reference data). These reference models are then used to estimate the sample distribution and select informative samples from the public dataset. The victim black-box model is queried using the selected samples, and the returned results are used to train the substitute model, which is intended to have the same functionality as the black-box model.

We evaluated the proposed method on different victim models, and the results demonstrated that SwiftTheft is superior to the state-of-the-art approaches [1], [3], [9] in terms of *agreement* (i.e., similarity) and time consumption in each iterative process. Furthermore, we carried out a quantitative analysis to explore the impact of reference data, the number of reference models on the performance of SwiftTheft. We also evaluated the time cost to confirm that SwiftTheft can reduce the time consumption by more than 98%.

In summary, this paper elaborates on the following contribution.

- To the best of our knowledge, we are the first to utilize reference models to model extraction attacks. Unlike other methods that require multiple iterations of sample selection and substitute updates, SwiftTheft only needs to select the most informative samples once, eliminating the need for intermediate substitute model training. This significantly reduces time and computing resource costs.
- We design a fast distribution estimation algorithm to identify the most informative samples from the public data pool, which is more efficient and effective than the traditional active learning-based approach and Jacobian-based data augmentation methods. Furthermore, the selected query samples are model-agnostic and can be used to extract functionality from a variety of cloud models.
- Extensive experiments on a range of tasks demonstrate that SwiftTheft can achieve a higher level of agreement while reducing the sample selection time by an average of 98%.

II. Preliminaries

1. Problem formulation

Machine-Learning-as-a-Service is a bundle of cloud computing services that grant eligible end-users access to proprietary machine learning models that offer machine learning solutions containing model training, data transformations, and predictive analytics. The eligible users are charged on-demand or purchase a monthly or annual subscription to access the models. These models are trained on proprietary data, either collected by cloud platforms themselves or third-party data providers who share profits with cloud platforms. The training process of these models is also time-consuming and costs valuable computing resources, which sums up as expense of cloud providers. The end-user can access the service by directly uploading an image to get a result or writing a custom application to query images via API. However, due to the black box property of API, end users cannot access the specific detail of the victim models or the training information.

Nowadays, model stealing attacks have been studied involving various aspects: parameters stealing [5], hyperparameters stealing [6], architecture extraction [7], and functionality extraction [2], [3], [8]. In this paper, our goal is to steal the functionality of the backbox deep neural networks independent of their internals. Compared to the MLaaS service, there are also emerging needs for cloud vendors to train the model for end-user to deploy on the IoT devices. In this scenario, MonoCNN [10] has been proposed to reduce the model footprint while retaining model robustness. Model extraction attack mainly aims at stealing model with only black-box access, while in MonoCNN, model parameters are accessible to the user since the model parameters are reconstructed in the IoT devices.

Functionality extraction attack In this paper, we formulate the task as follows: the attacker is given the access to victim model $F_V : \mathcal{X} \to \mathcal{Y}$, where only inputing x and corresponding result y is available to the attacker. An adversary aims to extract a substitute model F_A with near-identical predicting performance as victim model F_V (e.g., outputing the same label as victim model within the problem domain input). The adversary has the ability to collect a systhetic training set $D_T = \{(x, F_V(x)\}$ where $x \sim P_A(X)$ is most informative data in terms of functionality extraction. To better illustrate the typical model extraction against black-box models of MLaaS, we depicted such a process in Figure 1.

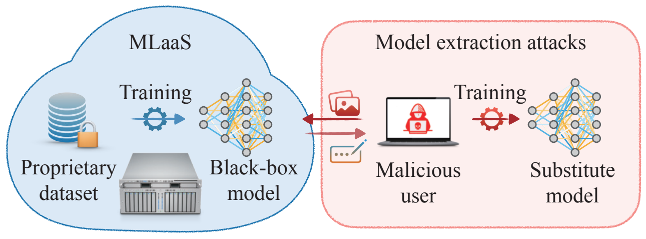


Figure 1 Model extraction attack against MLaaS provider.

2. Threat model

Following the mainstream MLaaS settings[11], we consider the adversary targets at the cloud-based neural network model, which is encapsulated and protected by cloud API, thus formulating the black-box scenario. In the black-box scenario, A M-dimensional confidence vector $\boldsymbol{y} = [y_1, y_2, \dots, y_M] \in \mathbb{R}^M$ can be acquired with arbitrary input $\boldsymbol{x}$ within the victim input space $\mathcal{X}$. In the toughest situation, the API only return the top-1 class label, which is the attacker's assumption in this paper. Moreover, we consider the most realistic scenario where adversary cannot learn any internal model information, including model parameters, model architecture, training information, or training dataset. While most information unknown, the adversary can obtain all the output class names within the service domain by enquiring the API. In terms of data availability, the adversary only has publicly available data and the ImageNet pre-trained models [12], which are involved as an inherent part of the major machine learning libraries, such as PyTorch and Tensorflow.

With the limited query budget and time, the adversary aims to functionally steal the victim model from the cloud provider and construct a similar performant substitute model as victim model. Without accessing the training dataset, SwiftTheft solely relies on non-problem domain natural data, e.g., the ImageNet64 dataset. SwiftTheft focus on budget-limited scenarios, and the query upper bound is the case when the attacker uses the total ImageNet64 dataset. Specifically, for CIFAR-10 and GTSRB models, the upper bound agreements of the substitute models are 84.99% and 93.68%, respectively.

3. Related works

1) Model extraction attack

In model extraction attacks, the adversary aims to train a substitute model with the same functionality as the victim model. According to the source of the query sample, the existing model extraction attacks can be categorized into three categories: Natural sample-based model extraction attacks, Synthesized sample-based model extraction attacks, and hybrid sample-based model extraction attacks.

i) Natural sample-based model extraction attacks

Natural samples are the samples downloaded from the public dataset, such as ImageNet and CIFAR-100. The public dataset can either be the problem domain (PD) or non-problem domain (NPD). The problem domain means that the samples belong to the distribution of the victim model training dataset. However, PD data is hard to acquire, especially the privacy-related fields, such as medical data. To recap, Correia-Silva *et al.* [9] proposed Copycat CNN that randomly selects query samples from the natural sample pool to train the substitute model. Orekondy *et al.* [2] proposed Knockoff nets that used an adaptive strategy to select the samples from the natural sample pool. The adaptive strategy is based on the feedback reward regarding sample efficiency, diversity, and training information. Chandrasekaran *et al.* [13] adopted extended adaptive training (EAT) to select samples with the least confidence scores. Pal *et al.* [1] proposed ActiveThief that combines DeepFool-based Active Learning (DFAL) strategy [14] to pick up the samples. Such samples are deemed to be easily perturbed and less redundant.

However, the above methods always have a trade-off between time cost and model performance.

ii) Synthesized sample-based model extraction attacks

Researchers also found that synthesized samples can also conduct model extraction attacks and explore input space unknown to natural samples. Up to now, several methods have been proposed to produce synthesized query samples. Tramèr *et al.* [5] first synthesized some random samples and then found the middle points between arbitrary two synthetic samples by binary search to synthesize the new samples. Maze [15] utilized a generator to generate samples that maximize the disagreement between the temporary substitute model and the victim model to improve the efficiency of model extraction. Data-Free Model Extraction (DFME) [16] maximizes the difference between the victim model and the substitute model by using surrogate data generated through a synthetic method. By finding the difference between the victim and substitute models and using this information to train the substitute model, DFME aims to minimize this difference and improve the attack performance. However, synthesized-based approaches suffer from an enormous query budget. For example, DFME [16] expends 20 million adversarial queries on the CIFAR-10 dataset when performing model extraction. In real-world scenarios, the high cost is a detriment to the commercial value and viability of such attacks.

iii) Hybrid sample-based model extraction attacks

Based on the benefit of both natural and synthetic samples, researchers have proposed hybrid sample-based model extraction attacks in recent years. Jacobin-based augmentation attacks (JBA) [3] are typical model extraction attacks using hybrid samples. In JBA, it is assumed that the adversary can access a subset of the victim's training data and use them to generate adversarial examples to train the substitute model. Juuti *et al.* [17] adopted the iterative fast gradient sign method (I-FGSM) [18] with a targeted randomly chosen direction (T-RND) in every step to reduce synthetic overlap. Yu *et al.* [8] proposed CloudLeak that used margin-based adversarial active learning [8] to generate synthesized datasets. Gong *et al.* [19] provided InverseNet, which makes use of a temporary substitute model to select samples of high confidence scores for producing high-quality inversed datasets.

However, hybrid-based methods have different caveats as well. JBA-based methods not only suffer marginal effects [20], but also require a subset of the victim dataset to bootstrap the attack process. Cloudleak [8] requires guide images, which violates the black-box setting of the model extraction attack. While effective, INVERSENET [19] requires burdensome inverse model training, which consumes a large number of computing resources and time.

2) Active learning

An accurate model needs to be trained on a large data pool in the deep learning scenario, thus requiring considerable efforts to label the data. Active learning employs an oracle to label samples, producing the desired output, which is similar to model extraction. However, active learning usually focuses on white-box scenarios, where the source knowledge data (victim data in model extraction scenario) is exposed to the model trainer. Depending on sampling strategies, active learning can be classified into membership query synthesis, stream-based selective sampling, and pool-based sampling.

i) Membership query synthesis

This method aims to build efficient training sets by synthesizing the most informative samples. Schumann *et al.* [21] approximated the decision boundary by the binary search and then picked up a random vector orthogonal to the mid-perpendicular vector of the decision boundary to help generate new samples. Yan *et al.* [22] provided a method iteratively producing a certain number of synthesized samples, from which samples will be selected for query in consideration of uncertainty, diversity, and representativeness for real data.

ii) Stream-based selective sampling

It's convenient to synthesize the most informative samples, but the synthesized data is sometimes meaningless and hard to label. Considering this fact, selecting the unlabeled natural data by stream-based selective sampling methods may be suitable. In these methods, samples are observed in real-time, and the adversaries determine whether to query the samples. Hong *et al.* [23] introduced a selection criterion that avoids unnecessary queries employing the preliminary substitute model.

iii) Pool-based sampling

The stream-based selective sampling methods are based on the assumption that adversaries can observe samples in real-time. However, in the case that adversaries can get access to a pool of unlabeled samples, pool-based sampling methods will be more advisable, in which adversaries use sampling strategies to sample from the pool. Mayer *et al.* [24] utilized GAN to synthesize high entropy samples and a discriminator to make sure the synthesized samples are indistinguishable from natural data so that adversaries can use a feature extractor to seek the most similar natural samples from the pool. Zhang *et al.* [25] made use of the unified representation generator to learn the representation of image samples and embed annotations into the representation, employing the most informative unlabeled samples.

III. Methodology of SwiftTheft

1. Overview

In this paper, we proposed a time-efficient active-based model extraction framework SwiftTheft, depicted in Figure 2. The process mainly consists of three different stages:

- Attack initialization. The attacker first selects and initializes a set of reference models as feature space mapping oracles. These models are then used to map the public data pool to a feature space. The attacker also selects seed samples as the initial selected set.
- Samples selection. The attacker then uses the distribution calculation algorithm to estimate the distribution of the selected sample pool and selects the most informative samples for model training.
- Substitute model training. The attacker then utilizes the selected samples to query the victim model. The substitute model is trained based on the query results to achieve high similarity with the victim model.

We detail the attack process of SwiftTheft in the Algorithm 1. Our experiments show that SwiftTheft can significantly reduce the sample selection time while achieving a better performance than ActiveThief [1] and Papernot *et al.*'s strategy [3].

2. Attack initialization

We preprocess the dataset pool in two steps in the attack initialization process. Firstly, the dataset pool will be mapped to the feature space. Next, we randomly select some seed samples from the dataset pool to prepare for subsequent sample selection.

For the sample mapping process, unlike the existing works [1], [5], [17], [19] that use an initial substitute model to map the public data pool to feature space, we initialize the reference model $F_P : X \to Z$ on the public dataset, where $X \in \mathcal{X}$, and $Z \in \mathbb{R}^f$. $\mathcal{X}$ is the input space of the reference model, and f is the feature dimension of the reference model. Note that the feature dimension of the reference model is identical to the feature dimension

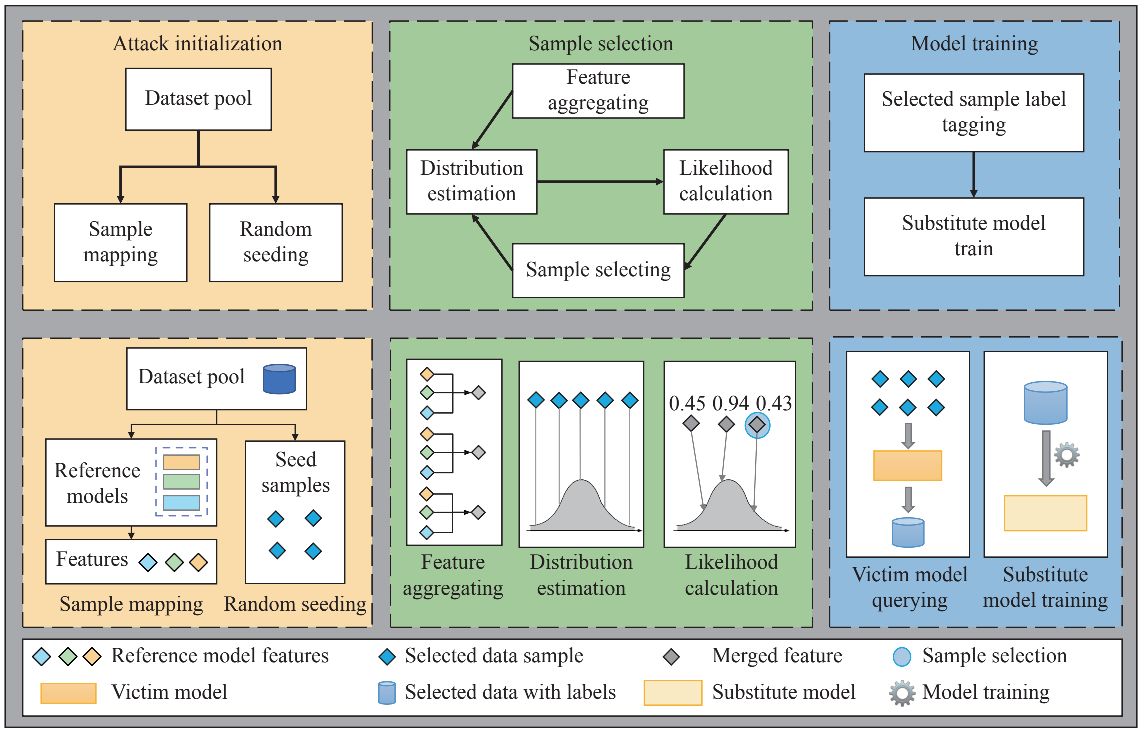


Figure 2 Overview of SwiftTheft.

of the number of the reference dataset categories.

Given the reference model, we map each data sample to feature space to form a feature pool $\tilde{D}$, which can be formulated as follows:

$$\tilde{D} = \{x_n, F_P(x_n)\} \tag{1}$$

A single reference model can provide distribution information of a data sample. However, due to the training settings of the reference model and the underlying model architecture, the distribution output of a single reference model will be biased. So, we use multiple reference models to alleviate distribution bias.

Algorithm 1 SwiftTheft

```
Data: Public data pool $D, D_R$; Aggregated features $\hat{D}$.
  Initial features $\hat{D}_j$ for $j$ in $1:m$.
// Attack Initialization
 1: for $j$ in $1:m$ do
 2:   $D_R^j = \text{Subset}(D_R, j)$;
 3:   Initialize model $F_R^j$ with random parameter $\Theta_j$;
 4:   Train model $(F_R^j, D_R^j)$;
 5: for $n$ in $S$ do
 6:   for $j$ in $1:m$ do
 7:     $y_n^j = F_R^j(x_n)$;
 8:     $\hat{D}_j = \hat{D}_j \cup \{y_n^j\}$;
 9: $S_0 = \text{Random}(S, s_0)$;
// Sample Selection
10: for $n$ in $S$ do
11:     $z_n = \sum_{j=0}^{m} y_n^j$;
12:     $\hat{D} = \hat{D} \cup \{x_n, z_n\}$;
13: for $i$ in $0:I$ do
14:     $\mu_i = \frac{1}{|S_i|}\sum_{n\in S_i} z_n$;
15:     $\sigma_i^2 = \frac{1}{|S_i|}\sum_{n\in S_i}(z_n - \mu_i)^2$;
16:     $\tilde{S}_{1,i} = S/S_i$; //$\tilde{S}_{1,i}$ refers to unselected samples
17:     $S_{i+1} = S_i$;
18:     for $k$ in $1:K$ do
19:         $n_{k,i} = \arg_n \min_{n\in\tilde{S}_{k,i}} \log p_i(z_n)$;
20:         $\tilde{S}_{k,i} = \tilde{S}_{k-1,i}/\{n_{k,i}\}$;
21:         $S_{i+1} = S_{i+1} \cup \{n_{k,i}\}$;
22: $D_I = x_n, n \in S_I$; // Model training
23: $D_T = \text{query}(F_V, D_I)$;
24: Train substitute model$(F_A, D_T)$.
```

Reference model training The attacker initializes m reference models $F_R^j : X \rightarrow Z$. Each reference model is trained on the subset of reference data D_R. Reference data helps the reference models learn about the distribution information of the public dataset. The reference data is an arbitrary public dataset with no overlap with the victim training data set. Therefore, the reference model can be obtained by retraining an off-the-shelf pre-trained model. We split the reference data into multiple non-overlapping data subsets D_R^j, and each of the subsets is used to train a reference model F_R^j. Note that the reference dataset has no overlap with the training dataset of the victim model, i.e., $D_R^j \cap D_V = \emptyset \quad \forall j$, where D_V is the training dataset of the victim model. We initialize different reference models by varying the parameter initialization settings. It is time-consuming to train a large amount of reference models, however, reference models are model-agnostic, which means reference models can transfer to multiple attack processes with one time training.

Feature mapping process We aim to map each data sample within the public data pool to the feature space in the feature mapping process. The number of the obtained feature is related to the number of the reference model. Given a data sample, one reference model will output a unique feature $F_R^j(x)$. We denote all the features generated by reference models F_R^j as $\hat{D}_j$.

$$\hat{D}_j = \{F_R^j(x_n) \colon \forall x_n \in D\} \quad \text{for } j = 1, 2, \ldots, m \tag{2}$$

Seed sample selection We randomly select a small number of seed samples from the public dataset D and preserve them as the selected sample pool to bootstrap the distribution estimation, which will be elaborated in Section. We denote the index set of the whole public dataset D as S. And the index set of the randomly selected seed samples is denoted as S_0.

3. Sample selection

Feature aggregating The first step in the sample selection process is feature aggregating, which aggregates all mapped features into a single feature vector for the given sample. It is based on an intuition that each feature vector generated by the reference model may be biased due to the training process and the reference data distribution. By feature aggregating, the bias in each feature can be reduced, thus making the distribution estimation of the selected sample pool more accurate. In feature aggregating, each reference model is considered to have equal weight in the aggregated feature. Consider that each data sample x_n has m feature vectors, the final vector is the sum of each feature $F_R^i(x_n)$. Thus the aggregated feature pool is calculated as

$$\hat{D} = \left\{ x_n, \sum_{j=0}^{m} F_R^j(x_n) \right\} \tag{3}$$

To simplify the notation, the aggregated feature for each sample is denoted as z_n, where n is the index of each sample.

Distribution estimation Given the aggregated feature, SwiftTheft iteratively selects the most informant samples. The sample selection iteration starts with distribution estimation for the selected samples. The seed samples are chosen as selected samples, whose index set is denoted as S_0, to bootstrap the iterative selecting process. For iteration i, the distribution estimation is conducted on the selected sample pool of the previous iteration, which denotes S_i. Note that for the first iteration, the selected samples are the seed samples S_0, and the iteration number starts from 0. The selected sample features z_n are assumed to follow normal distribution $\mathcal{N}(\mu, \Sigma)$. To estimate of the feature distribution, we comply with the common settings, which calculates mean and variance as

$$\mu_i = \frac{1}{|S_i|} \sum_{n \in S_i} z_n, \quad \sigma_i^2 = \frac{1}{|S_i|} \sum_{n \in S_i} (z_n - \mu_i)^2 \tag{4}$$

In formula (4), S_i is the index of the selected sample pool in the i-th iteration, and $|S_i|$ refers to the quantity of the S_i. The calculation of the variance σ_i depends on the result of μ_i.

Likelihood calculation Based on the distribution of the selected samples, the attacker can calculate the likelihood of the remaining samples belonging to the distribution, which can be formulated as

$$p(z) = \frac{1}{\sqrt{2\pi^k |\Sigma|}} \exp\left\{ -\frac{1}{2} (z - \mu)^\top \Sigma^{-1} (z - \mu) \right\} \tag{5}$$

The goal of sample selection is to obtain more diverse query samples. In general, if a sample has been selected before, the probability of belonging to the selected sample distribution will be higher. Thus, we will not choose the samples with a significant likelihood in the current iteration. Given the mean and variance of the selected samples, we can theoretically calculate the likelihood of each sample belonging to the selected sample distribution by applying equation (5). However, the probability density function of the multivariate normal distribution is hard to tackle. In this paper, we assume each feature is independent of the other so that the co-variance of Σ equals 0. We further use the log-likelihood to replace the burdensome exponential calculation. Thus, the negative log-likelihood of the sample belonging to distribution can be simplified as

$$\log p_i(z) = -\sum_{n=1}^{f} \frac{\|z_n - \mu_i\|}{\sigma_i} + \text{const} \tag{6}$$

The constant in the log probability equation will not affect the comparison procedure, so the constant term const is omitted in the log probability calculation. For each iteration i, with the μ_i and σ_i, each sample can be assigned with a log probability.

Query sample selection The K samples that are least likely to follow the selected pool distribution are added to the chosen sample pool for the next distribution estimation to increase the sample diversity. Formally,

$$n_{k,i} = \arg_n \min_{n \in \tilde{S}_{k,i}} \log p_i(z_n), \quad k \in [1, K] \tag{7}$$

where $\tilde{S}_{1,i} = S/S_i$ and $\tilde{S}_{k,i} = \tilde{S}_{k-1,i}/\{n_{k,i}\}$, $k \in [2, K]$, the selected pool is then updated as $S_{i+1} = S_i \cup \{n_{k,i}\}_{k=1}^{K}$.

After I iterations, when the selected sample pool S_I satisfies the target query budget B, we obtain the query set $D_I = \{x_n\}, n \in S_I$.

4. Substitute model training

We initialize the substitute model with random parameters Θ_{F_A}, and it has the same number of labels as the victim model. The attacker queries the black-box model with the selected samples D_I and obtain their labels. Then the attacker can construct the substitute model training dataset, which consists of selected samples and corresponding labels, which is $D_T = \{x_n, F_V(x_n)\}$, $x_n \in D_I$. After retraining the initialized substitute model on D_T, the attacker can get the final substitute model.

IV. Evaluation

1. Experiment Setup

We evaluate SwiftTheft on six different datasets, and the details of the datasets are shown in the Table 1.

Table 1 Summary of the related datasets

Dataset	VGG-Flower	CIFAR10	GTSRB	CIFAR100	SHVN	ImageNet64
Classes	10	10	43	100	10	1,000
Total samples	1,873	60,000	61,839	60,000	60,000	178,116
Training samples	1,673	50,000	39,209	50,000	50,000	128,116
Testing samples	200	10,000	12,630	10,000	10,000	50,000
Image size	Arbitrary	32 × 32	Arbitrary	32 × 32	32 × 32	64 × 64
Complexity	Moderate	Moderate	Moderate	Complex	Simple	Complex
Victim model	√	√	√	–	–	–
Reference model	√	√	√	√	√	√
Substitute model	–	–	–	–	–	√

• Victim model training. We train three victim models on GTSRB, CIFAR-10, and VGG-Flower dataset, respectively. Following ActiveThief [1] and InverseNet [19], we use CNN32 for the victim models. As for training hyper-parameter, we use SGD with momentum as the optimizer, with a learning rate of 0.01 and a momentum of 0.5. We train each victim model for 300 epochs.

• Black-box querying. We use ImageNet64 [12] as the natural data pool to query the black box and get the corresponding result. The query data, together with the query result, are then utilized for training the substitute model, which is functionally similar to the black box. To attest the efficiency of SwiftTheft, only the selected samples are queried.

• Reference model training. We train the reference model on ImageNet64 with ResNet-34 architecture for comparison with other SOTA methods. For the ablation study on the reference model, we selected three datasets with varying complexity, including SVHN, CIFAR-100, and ImageNet64. We also include three victim datasets in the ablation study, which are the upper bound of reference model settings. For training hyper-parameters, the optimizer is set to ADAM with a learning rate of 0.01, and each reference model is trained for 100 epochs.

• Substitute model training. As for the final training process detailed in Section III.4, we trained the model using CNN32 and the same training hyper-parameters as the victim models. The training data of the substitute model is the query result of the victim model alongside selected query data.

Following evaluation method in various other works [1], [5], [17], [26], we adopt *agreement* as the evaluation metric. Formally,

$$\text{agreement}(F_A, F_V) = \frac{1}{|D_T|} \sum_{x \in D_T} \mathbb{1}(F_A(x), F_V(x)) \quad (8)$$

where $\mathcal{D}_T$ is the query dataset, $\mathbb{1}$ refers to the comparing function that outputs 1 when the labels are the same and 0 otherwise. *Agreement* is used to measure the similarity degree between the black-box model and the substitute model.

All the experiments are conducted on an Intel Xeon CPU, 64GB of RAM server with NVIDIA GTX 2080 GPU, which runs Ubuntu 18.04 and PyTorch 1.8.

2. State-of-the-art model extraction baselines

We compare SwiftTheft with three state-of-the-art model extraction attacks, i.e., Copycat CNN [9], ActiveThief [1], and Papernot *et al.*'s strategy [3]. We run these attacks based on their open-source codes.

• Copycat CNN. Copycat CNN [9] uses natural samples (publicly available data) to conduct the attacks. In each iteration, Copycat CNN randomly selects a set of samples from the data pool to query the victim model.

• ActiveThief. Unlike Copycat CNN, ActiveThief [1] utilizes active learning (e.g., K-Center) algorithm to select query samples from the natural sample pool. We randomly select 500 samples as the initial seed data and select 500 samples in each iteration.

• Papernot *et al.*'s strategy. Unlike the above methods that use natural samples, Papernot *et al.*'s strategy [3] uses the Jabocabian-based method [3] to generate query samples based on a small number of seed samples. In this paper, we use a subset of ImageNet64 as seed samples. Following Papernot *et al.*'s strategy [3], the augmentation algorithm is the fast gradient sign method (FGSM) [27].

3. Quantitative analysis

Comparison with state-of-the-art attacks The comparison results are shown in Figure 3. To make a fair comparison, we use the same query budget and experimental settings for both baselines and SwiftTheft. We can see that SwiftTheft outperforms all the baseline methods at all budgets. Notably, the performance gap between SwiftTheft and baseline attacks is most pronounced when the query budget is limited. For example, SwiftTheft achieves an agreement of 73.73%, 64.39%, and 83.00% on CIFAR10, GTSRB, and VGG-Flower at query budgets of 4000. In comparison ActiveThief only achieves an agreement of 60.04% (CIFAR10), 62.36% (GTSRB), and 80.50% (VGG-Flower), followed by CopyCat CNN, which only reaches agreement of 59.40% (CIFAR10), 59.24% (GTSRB), and 70.50% (VGG-Flower). Surprisingly, in such cases, Papernot *et al.*'s strategy only reaches an agreement of 30.30%, 47.13%, and 56.50% on CIFAR10, GTSRB, and VGG-Flower. Compared to CopyCat CNN and ActiveThief, the success of SwiftTheft is due to the distribution estimation algorithm and introduction of reference model settings. We attribute the reason why Papernot *et al.*'s strategy has the worst attack performance to the insufficient natural data samples and the marginal effect [20] of synthetic data.

Impact of the reference data In this part, we investigate the impact of reference model training data on the performance of SwiftTheft. We choose six different datasets as reference data. We also use the training datasets as the reference datasets as the theoretical upper bound. The experimental results are shown in Table 2.

It is shown in Table 2 that when the reference data is identical to the victim training data, the attacker can obtain the highest agreement. Besides, we discover that the attacker can obtain a satisfactory attack result with a more complex dataset, such as the ImageNet64 and CIFAR100.

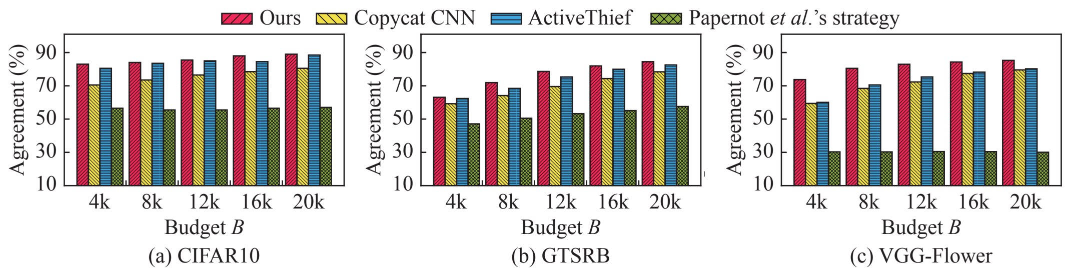


Figure 3 Comparison with Copycat CNN [9], ActiveThief [1], and Papernot *et al.*'s strategy [3].

Table 2 Impact of reference data

Dataset	Reference data	Budget				
		4k	8k	12k	16k	20k
VGG-Flower	SVHN	83.50%	87.00%	89.50%	90.00%	89.50%
	VGG-Flower	85.00%	**91.00%**	**91.00%**	**91.00%**	**91.50%**
	GTSRB	82.50%	86.50%	87.50%	89.00%	89.00%
	CIFAR10	84.50%	88.50%	99.50%	88.50%	90.00%
	CIFAR100	**86.00%**	89.00%	89.00%	90.50%	90.50%
	ImageNet64	**86.00%**	86.50%	88.50%	88.50%	89.00%
CIFAR-10	SVHN	52.04%	66.56%	72.95%	74.24%	78.24%
	VGG-Flower	58.35%	62.40%	76.11%	77.82%	76.06%
	GTSRB	58.26%	70.09%	76.39%	78.23%	80.88%
	CIFAR10	**63.06%**	71.07%	**78.35%**	**80.34%**	**81.36%**
	CIFAR100	59.19%	68.82%	74.69%	78.15%	79.04%
	ImageNet64	61.17%	**71.45%**	74.06%	77.89%	80.67%
GTSRB	SVHN	59.18%	66.12%	72.11%	77.24%	80.44%
	VGG-Flower	60.16%	64.95%	71.27%	74.41%	75.28%
	GTSRB	**64.13%**	**72.30%**	**78.27%**	**82.29%**	83.79%
	CIFAR10	60.19%	64.64%	74.85%	76.29%	80.78%
	CIFAR100	63.55%	65.26%	72.69%	77.13%	81.25%
	ImageNet64	63.08%	71.92%	78.61%	81.96%	**84.49%**

Analysis of the time cost In this part, we make a comparison of the time cost of SwiftTheft with that of the baseline attacks on an iteration basis. Since Copycat CNN is a one-time process, we only compare SwiftTheft with the other baselines. The comparison begins from the second iteration since both SwiftTheft and baselines have a similar bootstrap process, i.e., selecting a small number of seed samples.

Unlike ActiveThief and Papernot *et al.*'s strategy that have different time consumption on different datasets, SwiftTheft is a model-agnostic approach. Hence, SwiftTheft can be applied to various victim models while performing only one data sample selection process. In Table 3, each data point is the iteration duration under the currently selected data pool. Compared with baselines, SwiftTheft can drastically reduce the time consumption of each iteration. In addition, as the selected samples increase, the time consumption of SwiftTheft can still remain basically unchanged while ActiveThief is increasing rapidly.

Table 3 Time (S) of selecting or synthesizing current batch of data

Selected samples	GTSRB		CIFAR10		VGG-Flower		SwiftTheft
	ActiveThief [1]	Papernot *et al.*'s strategy [3]	ActiveThief [1]	Papernot *et al.*'s strategy [3]	ActiveThief [1]	Papernot *et al.*'s strategy [3]	
1000	12620.57	639.30	12610.75	671.31	9364.95	554.48	10.76
1500	3939.94	719.38	3759.87	603.90	3324.62	566.08	10.72
2000	1933.33	714.60	1847.64	633.37	3640.37	580.50	10.37
2500	2101.41	781.83	1947.76	634.45	1858.98	599.27	10.27
3000	2482.72	784.07	1887.68	629.94	1773.27	606.12	11.57
3500	2665.98	784.88	2194.64	701.93	2325.85	600.94	11.39
4000	3023.09	793.77	2685.04	662.39	2469.56	631.78	10.44
4500	3219.33	817.66	2990.38	720.73	2782.97	638.84	9.90
5000	3691.46	835.25	3480.07	760.72	3324.04	623.84	10.51
5500	4077.90	839.87	3733.92	777.88	3538.48	638.77	10.19
6000	4547.31	891.91	3865.91	763.12	3942.44	655.19	11.31
6500	5064.37	928.03	4391.21	822.92	4330.01	692.54	10.19
7000	5169.38	937.40	4580.82	863.89	4530.90	688.15	10.20
7500	5468.61	981.68	4966.97	866.63	4941.78	712.22	9.91
8000	6216.50	901.06	5416.10	891.06	5170.28	747.46	9.70
8500	9136.74	1024.50	5862.50	913.19	5774.74	767.98	10.30
9000	18195.98	924.67	8708.38	903.23	6097.57	788.22	10.27
9500	11265.57	882.24	4679.12	930.17	14407.71	793.95	10.09
10000	6503.59	830.64	5192.82	854.20	7342.95	824.75	10.34

Impact of the number of reference models The main intuition of using multiple reference models is that combining feature outputs with different models on the same dataset can reduce the bias of the models. To demonstrate the effectiveness of multiple reference models, we vary the reference model number and test the agreement of the substitute model. The results are shown in Table 4.

Table 4 Impact of the number of reference models

Dataset	Model number	Budget				
		4k	8k	12k	16k	20k
VGG-Flower	1	84.00%	85.50%	88.00%	88.00%	88.00%
	8	85.50%	86.00%	88.00%	**88.50%**	**89.50%**
	16	**86.00%**	**86.50%**	**88.50%**	**88.50%**	89.00%
CIFAR-10	1	60.50%	71.05%	75.46%	78.08%	79.10%
	8	61.16%	70.19%	**76.12%**	**78.67%**	79.54%
	16	**61.17%**	**71.45%**	74.06%	77.89%	**80.67%**
GTSRB	1	60.34%	70.71%	**80.02%**	82.35%	84.12%
	8	60.62%	70.72%	79.82%	**82.87%**	84.28%
	16	**63.08%**	**71.92%**	78.61%	81.96%	**84.49%**

When the query budget is under 12k, we can see that the agreement increases as the reference model increases. When the budget is above 12k, the agreement will fluctuate slightly up and down. It is because SwiftTheft suffers from the marginal effect when the query budgets are too immense. In general, using multiple reference models will lead to better and more stable performance. Under the same training condition (i.e., training dataset, training parameters), the model with the same structure will output similar features. However, in our multiple reference model setting, reference models have different architectures. And combining features from multiple reference models provides a more stable feature set and better extraction performance. Furthermore, it is better to have different model architectures with multiple reference models.

V. Conclusion

This paper presents the design, implementation, and empirical analysis of an effective model extraction attack SwiftTheft. SwiftTheft uses reference models to select informant samples. Performance evaluation also reveals that the selecting result of SwiftTheft has a universal performance benefit across various victim models. Extensive experiments confirm that SwiftTheft is effective and efficient and presents as an outstanding method of model extraction attack than the current state-of-the-art approaches.

Acknowledgement

This work was partially supported by the National Key R&D Program of China (Grant No. 2020AAA0107701), the NSFC (Grant No. U20B2049 and U21B2018).

References

[1] S. Pal, Y. Gupta, A. Shukla, *et al.*, "ActiveThief: Model extraction using active learning and unannotated public data," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, New York, NY, USA, pp. 865–872, 2020.

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

[3] N. Papernot, P. McDaniel, I. Goodfellow, *et al.*, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, Abu Dhabi, United Arab Emirates, pp. 506–519, 2017.

[4] Q. X. Zhang, W. C. Ma, Y. J. Wang, *et al.*, "Backdoor Attacks on Image Classification Models in Deep Neural Networks," *Chinese Journal of Electronics*, vol. 31, no. 2, pp. 199–212, 2022.

[5] F. Tramèr, F. Zhang, A. Juels, *et al.*, "Stealing machine learning models via prediction APIs," in *Proceedings of the 25th USENIX Conference on Security Symposium*, Austin, TX, USA, pp. 601–618, 2016.

[6] B. H. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proceedings of 2018 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, pp. 36–52, 2018.

[7] V. Duddu, D. Samanta, D. V. Rao, *et al.*, "Stealing neural networks via timing side channels," *arXiv preprint*, arXiv: 1812.11720, 2018.

[8] H. G. Yu, K. C. Yang, T. Zhang, *et al.*, "CloudLeak: Large-scale deep learning models stealing through adversarial examples," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, 2020.

[9] J. R. Correia-Silva, R. F. Berriel, C. Badue, *et al.*, "Copycat CNN: Stealing knowledge by persuading confession with random non-labeled data," in *Proceedings of 2018 International Joint Conference on Neural Networks*, Rio de Janeiro, Brazil, pp. 1–8, 2018.

[10] C. Ding, Z. Lu, F. J. Xu, *et al* ., "Towards Transmission-Friendly and Robust CNN Models over Cloud and Device," *arXiv preprint*, arXiv: 2207.09616, 2022.

[11] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "MlaaS: Machine learning as a service," in *Proceedings of the IEEE 14th International Conference on Machine Learning and Applications*, Miami, FL, USA, pp. 896–902, 2015.

[12] M. Simon, E. Rodner, and J. Denzler, "ImageNet pre-trained models with batch normalization," *arXiv preprint* , arXiv: 1612.01452, 2016.

[13] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, *et al* ., "Exploring connections between active learning and model extraction," in *Proceedings of the 29th USENIX Security Symposium* , pp. 1309–1326, 2020.

[14] M. Ducoffe and F. Precioso, "Adversarial active learning for deep networks: A margin based approach," *arXiv preprint* , arXiv: 1802.09841, 2018.

[15] S. Kariyappa, A. Prakash, and M. K. Qureshi, "MAZE: Data-free model stealing attack using zeroth-order gradient estimation," in *Proceedings of* 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, pp. 13814–13823, 2021.

[16] J. B. Truong, P. Maini, R. J. Walls, *et al* ., "Data-free model extraction," in *Proceedings of 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Nashville, TN, USA, pp. 4771–4780, 2021.

[17] M. Juuti, S. Szyller, S. Marchal, *et al* ., "PRADA: Protecting against DNN model stealing attacks," in *Proceedings of 2019 IEEE European Symposium on Security and Privacy*, Stockholm, Sweden, pp. 512–527, 2019.

[18] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proceedings of the 5th International Conference on Learning Representations* , Toulon, France, 2015.

[19] X. L. Gong, Y. J. Chen, W. B. Yang, *et al* ., "InverseNet: Augmenting model extraction attacks with training data inversion," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence* , Montreal, Canada, pp. 2439–2447, 2021.

[20] Z. Y. Zhang, Y. Z. Chen, and D. A. Wagner, "Towards characterizing model extraction queries and how to detect them," EECS Department, University of California, *Technical Report* , No. UCB/EECS-2021-126, 2021.

[21] R. Schumann and I. Rehbein, "Active learning via membership query synthesis for semi-supervised sentence classification," in *Proceedings of the 23rd Conference on Computational Natural Language Learning*, Hong Kong, China, pp. 472–481, 2019.

[22] Y. F. Yan, S. J. Huang, S. Y. Chen, *et al.*, "Active learning with query generation for cost-effective text classification," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, New York, NY, USA, pp. 6583–6590, 2020.

[23] S. Hong and J. Chae, "Active learning with multiple kernels," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2980–2994, 2022.

[24] C. Mayer and R. Timofte, "Adversarial sampling for active

learning," in *Proceedings of 2020 IEEE Winter Conference on Applications of Computer Vision*, Snowmass, CO, USA, pp. 3071–3079, 2020.

[25] B. C. Zhang, L. Li, S. J. Yang, *et al*., "State-relabeling adversarial active learning," in *Proceedings of 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seattle, WA, USA, pp. 8756–8765, 2020.

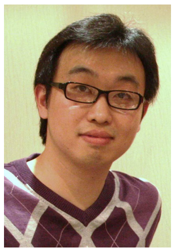
[26] M. Jagielski, N. Carlini, D. Berthelot, *et al* ., "High accuracy and high fidelity extraction of neural networks," in *Proceedings of the 29th USENIX Security Symposium*, Virtual event, pp. 1345–1362, 2020.

[27] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, CA, USA, arXiv: 1412.6572, 2015.

Wenbin YANG was born in 1997. He received the B.S. degree from School of Cyber Science and Engineering at Wuhan University, in 2020. He is currently pursuing the M.S. degree in School of Cyber Science and Engineering at Wuhan University, China. His research interests include machine learning and deep learning security.
(Email: yangwenbin@whu.edu.cn)

Xueluan GONG was born in 1996. She received the B.S. degree in computer science and electronic engineering from Hunan University in 2018. She is currently pursuing the Ph.D. degree in the School of Computer Science, Wuhan University, China. Her research interests include network security and AI security. (Email: xueluangong@whu.edu.cn)

Yanjiao CHEN received the B.E. degree in electronic engineering from Tsinghua University in 2010 and Ph.D. degree in computer science and engineering from Hong Kong University of Science and Technology in 2015. She is currently a Bairen Researcher in Zhejiang University, China. Her research interests include computer networks, wireless system security, and network economy. She is a Member of the IEEE.



Qian WANG was born in 1980. He received the Ph.D. degree from the Illinois Institute of Technology, USA. He is a Professor at the School of Cyber Science and Engineering, Wuhan University, China. His research interests include AI security, data storage, search and computation outsourcing security, etc.

Prof. Wang received the National Science Fund for Excellent Young Scholars of China in 2018. He is a recipient of the 2016 IEEE Asia-Pacific Outstanding Young Researcher Award. He serves as Associate Editors for *IEEE Transactions on Dependable and Secure Computing (TDSC)* and *IEEE Transactions on Information Forensics and Security (TIFS)*. (Email: qianwang@whu.edu.cn)

Jianshuo DONG is currently an undergraduate at the School of Cyber Science and Engineering in Wuhan University, China. His research interests include machine learning and deep learning security.