MetaSimulator: Simulating Unknown Target Models for Query-Efficient Black-box Attacks

Chen Ma, Li Chen and Junhai Yong
School of Software, BNRist, Tsinghua University
Beijing 100084, P. R. China
machanic@126.com, {chenlee, yongjh}@tsinghua.edu.cn

Abstract

Many adversarial attacks have been proposed to investigate the security issues of deep neural networks. For the black-box setting, current model stealing attacks train a substitute model to counterfeite the functionality of the target model. However, the training requires querying the target model. Consequently, the query complexity remains high and such attacks can be defended easily by deploying the defense mechanism. In this study, we aim to learn a generalized substitute model called MetaSimulator that can mimic the functionality of the unknown target models. To this end, we build the training data with the form of multi-tasks by collecting query sequences generated in the attack of various existing networks. The learning consists of a double-network framework, including the task-specific network and MetaSimulator network, to learn the general simulation capability. Specifically, the task-specific network computes each task’s meta-gradient, which is further accumulated from multiple tasks to update MetaSimulator to improve generalization. When attacking a target model that is unseen in training, the trained MetaSimulator can simulate its functionality accurately using its limited feedback. As a result, a large fraction of queries can be transferred to MetaSimulator in the attack, thereby reducing the high query complexity. Comprehensive experiments conducted on CIFAR-10, CIFAR-100, and TinyImageNet datasets demonstrate the proposed approach saves twice the number of queries on average compared with the baseline method. The source code is released on https://github.com/machanic/MetaSimulator.

1. Introduction

Deep neural networks (DNNs) are vulnerable to adversarial attacks [17, 50], which add human-imperceptible perturbations to the benign image to cause the target model to misclassify it. The study of adversarial attacks is crucial in identifying the weakness of DNNs [50, 4] and consequently contributes to the implementation of robust DNNs [37].

Adversarial attacks can be categorized into two main attacks, i.e., white-box and black-box attacks. In white-box attack setting, the target model is exposed fully to the adversary. Thus, the perturbation can be crafted easily using gradient information [37, 6, 17]. In black-box attack setting, the adversary has partial information of the target model, and crafts adversarial examples without any gradient information. The black-box attack setting is more practical in real-world systems. Many studies [46, 7, 13, 23, 5, 53, 25]...
have been proposed to perform black-box attacks and these methods can be divided into two categories, \textit{i.e.}, query-based attacks and transfer-based attacks.

Given that the gradient information cannot be obtained directly in the black-box setting, query-based attacks focus on the gradient estimation via zeroth-order optimizations \cite{7, 23, 25, 39}. These attacks turn out to be highly effective because of their satisfactory attack success rate. Despite the practical merit of query-based attacks, the high query complexity is inevitable for computing the high precision approximate gradient, making their procedures costly. In addition, the queries are typically underutilized, \textit{i.e.}, the implicit but profound messages returned from the target model are overlooked because they are abandoned after estimating the gradients. \textbf{How to make full use of the limited feedback of the target model to enhance the query-efficiency of attacks should be considered.}

Transfer-based attacks generate adversarial examples by using a standard white-box attack method on a pre-trained source model to fool the target model \cite{40, 33, 45, 58, 9, 42, 11, 22, 26, 51}. The major problems of the transfer-based attacks are that (1) they cannot achieve a high success rate; (2) they are weak in the targeted attack. To improve the transferability, model stealing attacks train a local substitute model to mimic the black-box model using a synthetic dataset, in which the labels are given by the target model through queries \cite{35, 52, 46, 43}. The difference between the substitute and the target model is minimized, and thus the attack success rate is increased. However, the training requires querying the target model. Consequently, the query complexity remains high and such attacks can be defended easily by deploying defense mechanisms (\textit{e.g.}, \cite{44, 30}). Besides, the inevitable re-training to substitute a new target model is expensive. \textbf{Hence, how to train a substitute model without the requirement of the target model is worth exploring.}

To eliminate the requirement of the target model during training, we propose a meta-learning-based approach to learn a generalized substitute model over many different networks, thereby exploiting their characteristics to achieve fast adaptation. The generalized substitute model named MetaSimulator can adapt to mimic the output of the target model which is unseen in training. Consequently, the large fraction of queries can be transferred to MetaSimulator. To this end, the proposed approach is equipped with a double-network framework to learn from tasks. Each task is a small data subset which is a query sequence generated during attacking a randomly selected pre-trained network. A large number of tasks allow MetaSimulator to experience the attacks of various networks, thereby assisting it in adapting to simulate the unknown target model rapidly. The gradients and meta-gradients of a batch of randomly sampled tasks are computed and finally aggregated by the optimization to update MetaSimulator to improve generalization. In the attack, the trained MetaSimulator will be further fine-tuned with the limited feedback of the unknown target model to simulate its output accurately, thereby transferring the query stress from the target model (Fig. 1). The proposed approach bridges the gap between model stealing attacks and the query-based ones: the query complexity of the attack is reduced by using MetaSimulator which is trained without using the target model. The elimination of the target model in training poses a new security threat in real-world systems: the attacker with the least amount of information about the target model can also mount a successful attack.

We evaluate the proposed method on CIFAR-10, CIFAR-100 \cite{29}, and TinyImageNet \cite{48} datasets and compare it with nature evolution strategy (NES) \cite{24}, Bandits \cite{25}, Meta Attack \cite{15}, random gradient-free (RGF) method \cite{9}, and prior-guided RGF (P-RGF) method \cite{9}. The results show the proposed approach saves twice the number of queries compared with the baseline method on average without sacrificing the attack success rate.

The main contributions are summarized as follows.

(1) We propose a novel black-box attack to bridge the gap between model stealing attacks and the query-based ones. Powered by meta-learning, the attacker can learn a substitute model to simulate the target model which is unseen in training for the first time.

(2) We propose to train a generalized substitute model called MetaSimulator on the queries collected from attacking various networks. Consequently, the query complexity of the attack is reduced by transferring a large fraction of queries to MetaSimulator.

(3) Extensive experiments on CIFAR-10, CIFAR-100, and TinyImageNet datasets demonstrate the superior query-efficiency of the proposed method.

2. Background

In this section, we introduce the backgrounds of black-box adversarial attacks and meta-learning.

2.1. Black-box Adversarial Attacks

Black-box attacks can be divided into two categories: query-based attacks and transfer-based attacks. Query-based attacks can be further divided into score-based attacks and decision-based attacks based on how much information returned from the target model that the adversary can utilize. Score-based attacks use the output probabilities (scores) of the target model to generate adversarial examples. Most score-based attacks estimate the approximate gradient through zeroth-order optimizations \cite{7, 23, 5, 15, 53}. Early studies \cite{7, 5} estimated the gradient via sampling from a noise distribution around the pixels, such
as zeroth order optimization attack (ZOO) [7], which uses a two-sided estimation to calculate the gradient of the $i$th pixel as follows:

$$
\hat{g}_i = \frac{L(x + he_i, y) - L(x - he_i, y)}{2h} \approx \frac{\partial L(x, y)}{\partial x_i},
$$

(1)

where $h$ is a parameter that controls the accuracy of the estimation and $e_i$ is the canonical basis vector ($e_i$ is 1 in the $i$th coordinate and 0 elsewhere). After obtaining the approximate gradient, the adversary optimizes the adversarial example iteratively with typical steps that follow the projected gradient descent attack (PGD) [37]:

$$
x_{t+1}^{adv} = \prod_{y} \mathcal{G}_p(x, y) (x_t^{adv} + \eta \cdot g_t),
$$

(2)

where $x_t^{adv}$ denotes the adversarial example optimized at step $t$, $x$ represents the original benign image, $\prod_{y} \mathcal{G}_p(x, y)$ is the operation that projects examples onto the $\ell_p$ ball centered at $x$ with the radius of $\epsilon$, $\eta$ denotes the learning rate, and $g_t$ is the estimated gradient. Although this type of approach can deliver a successful attack, it requires an extremely large number of queries to estimate the gradients in the high-resolution image because each pixel needs two queries. Methods are improved to reduce the search space by using the principal components of the data [5], searching perturbations in a latent space with reduced dimension [55], utilizing prior information about the gradient [25], utilizing evolutionary strategies [2], using random search [18, 3], searching the solution among the vertices of the $\ell_p$ ball [39], or adopting the Bayesian optimization [49]. In decision-based attacks, the attacker only knows the output label of the target model. Typical decision-based attacks include Boundary attack [14], $\mathcal{N}$ attack [31], Sign-OPT [8], etc. In this study, we focus on the score-based attack setting.

Transfer-based attacks generate adversarial examples on a source model and then transfer them to the target model [40, 33, 45, 42, 11, 22, 26, 51], or utilize the gradient information provided by a surrogate model [58, 9]. Transfer-based attacks cannot achieve a high attack success rate because of the difference between the source model and the target model. Many efforts have been devoted to improving the attack success rate (e.g., the use of an ensemble of source models [32]). Other efforts include model stealing attacks.

The original goal of model stealing attacks is to replicate the functionality of the public service by exploiting black-box access and no longer need to pay for the service [42, 54, 35, 52, 38, 10, 43]. This replication expands the scope of model stealing attacks. For example, Papernot et al. [45, 46] trained a substitute model to mimic the target model using a synthetic dataset labeled by the target model. Then, the local substitute is used to craft adversarial examples that can be transferred to the target model. This type of attack can be classified as a special type of transfer-based attack. In this study, we focus on enhancing this type of model stealing attack, i.e., how to train a substitute without using the target model. The terminologies of this study are as follows.

**Target model**: the victim black-box model of the attack.

**Untargeted attack**: the goal of this attack is to craft the adversarial examples that can be classified into any incorrect class.

**Targeted attack**: the goal of this attack is to craft the adversarial examples that can only be classified into the target class.

**Query pair**: the proposed attack uses the two-sided gradient estimation, in which a gradient is computed by integrating information between two consecutive queries.

**Evolving queries**: the queries that are updated iteratively during the attack process are evolving queries.

### 2.2. Meta-learning

Meta-learning is a learning-to-learn strategy that is useful in solving few-shot problems. In this strategy, the meta-learner can learn new skills or adapt to new environments rapidly with only a few samples. To achieve such generalization, typical meta-learning methods (e.g., model-agnostic meta-learning (MAML) [16]) are trained over a large number of tasks, which are defined as the small subsets of the original dataset, to allow the method optimize for the best performance over the task distribution, including potentially unseen tasks. In the testing phase, the meta-learner needs to be fine-tuned on a few labeled test samples to adapt to the test data. To utilize meta-learning in the adversarial attack field, Ma et al. [36] proposed MetaAdvDet to detect evolving adversarial attacks, which can recognize new types of adversarial examples with high accuracy. MetaAttack [15] trains an auto-encoder-based meta-learner on the gradients of multiple types of networks. After training, the gradients of the target model can be directly predicted by the meta-learner, thereby reducing the query complexity. MetaAttack is promising but it uses ZOO [7] to calculate approximate gradients for fine-tuning meta-learner, which yields a large number of queries and memory consumption. The prediction of the gradient map is also difficult for the auto-encoder, especially in the case of a high-resolution map. The proposed method uses the steps from Bandits attack to estimate gradients in the attack, which are more query-efficient than ZOO. In addition, MetaSimulator predicts the logits output rather than the gradient map, hence the performance is not affected by the resolution of input.
3. Method

3.1. Overview

The proposed approach consists of two stages: training stage for training MetaSimulator and attack stage that uses the trained MetaSimulator to reduce the query complexity of the attack. In the training stage (Algorithm 1), the MetaSimulator is trained on a large number of tasks. To construct the tasks, we collect the queries and corresponding outputs generated in attacking various networks and reorganize them as the form of multi-tasks (Fig. 2 left); each task only contains the data collected from one network. The purpose of introducing the data generated during the attack is to allow MetaSimulator to distinguish subtle differences of evolving queries. A large number of tasks create a huge simulation environment of various networks to improve the general simulation capability. The proposed approach is equipped with a double-network framework with the learning-to-learn strategy to learn from these tasks. This strategy learns how to simulate new models rapidly by reusing the experience of simulating the existing ones.

In the attack stage (Algorithm 2), the first phase is called “warm-up”, which uses a typical query-based attack algorithm (i.e., Bandits attack) to query the target model directly. However, the difference is that queries and the corresponding feedback of the target model are collected for fine-tuning the MetaSimulator. After fine-tuning, MetaSimulator is used as the substitute for the target model. Thus, subsequent k queries can be fed into MetaSimulator to alleviate the query stress of the target model. In addition, Algorithm 2 feeds queries to the target model and MetaSimulator alternately to refresh the simulation capability of the latter and keep up with the latest trend of evolving queries.

3.2. Task Construction

In the attack, MetaSimulator must simulate the outputs of any unknown target models accurately when the feeding queries are only slightly different from each other. To this end, MetaSimulator should learn from the real attack, i.e., the intermediate data (query sequences & outputs) generated in the attacks of various networks. Thus, a variety of pre-trained networks $\mathcal{M}_1, \ldots, \mathcal{M}_n$ are collected to construct the training tasks. The data inside each task come from the intermediate data generated by attacking one network using Bandits attack\(^1\). The data sources used by Bandits can be the training sets of the datasets, which have different data distribution from the tested images used in evaluation. The query pairs $Q_1, \ldots, Q_V$ generated from the randomly selected consecutive $V$ iterations of the attack are collected to form a task, which allows MetaSimulator to learn from the evolution of the real attack’s queries.

Besides, in the attack stage, the trained MetaSimulator must learn from the target model’s outputs generated in previous iterations to substitute the target model in subsequent iterations (Fig. 1). To this end, each task is divided into two subsets, namely, the meta-train set $\mathcal{D}_{\text{intr}}$, which is com-

\(^1\)Bandits attack uses the two-sided gradient estimation, where two perturbations (a query pair) are used to estimate a gradient.
posed of the first $t$ query pairs $Q_1, \ldots, Q_t$, and the meta-
test set $D_{mte}$ with the following query pairs $Q_{t+1}, \ldots, Q_V$.
The former is used in the inner update step of the training,
which corresponds to the fine-tuning step of the attack
stage. The fine-tuning step is also performed on the queries
which generated in previous iterations (Fig. 1), just like the
selection of $Q_1, \ldots, Q_t$ in this set. The meta-test set corre-
sponds to the subsequent $m$ query pairs of the attack stage
which are predicted by MetaSimulator; minimizing the test
error on this set is beneficial for improving the prediction
precision (the outer update step). Generally speaking, the
training stage and attack stage are connected seamlessly:
the partition of meta-train set and meta-test set in each task
(Fig. 2) matches two successive groups of attack iterations
(Fig. 1). The logits outputs of $M$ (Fig. 1), the partition of meta-train set and meta-test set in each task
(Fig. 2) matches two successive groups of attack iterations
(Fig. 1). The logits outputs of $M_1, \ldots, M_n$ are termed as
“pseudo labels”. All query sequences and pseudo labels are
pre-stored in the memory to accelerate the training.

Algorithm 1 The Training Procedure of MetaSimulator

Input: Training dataset $D$, Bandits attack algorithm $A$, pre-trained classification networks $M_1, \ldots, M_n$, MetaSimulator network $M$ and its parameters $M_0$, task-specific network $T$ and its parameters $T_0$, feed-forward function $f_{T_0}$ of $T$, loss function $\mathcal{L}(\cdot, \cdot)$ defined in Eqn. 3.

Parameters: Training iterations $N$, meta-train set size $t$, task size $V$, inner-update learning rate $\lambda_1$, outer-update learning rate $\lambda_2$, inner updates iteration $T$.

Output: The learned MetaSimulator $\hat{M}$

1: for iter $\leftarrow 1$ to $N$ do
2: sample $K$ benign images $x_1, \ldots, x_K$ from $D$
3: for $k \leftarrow 1$ to $K$ do
4: a network $M_k \leftarrow$ sample from $M_1, \ldots, M_n$
5: $Q_1, \ldots, Q_V \leftarrow A(x_k, M_k)$
6: queries generated from randomly selected consecutive $V$ attack iterations
7: $D_{mtr} \leftarrow Q_1, \ldots, Q_t$
8: $D_{mte} \leftarrow Q_{t+1}, \ldots, Q_V$
9: $P_{train} \leftarrow M_1(D_{mtr})$
10: $P_{test} \leftarrow M_1(D_{mte})$, $\triangleright$ pseudo labels
11: $T_0 \leftarrow T_0$, $\triangleright$ initialize $T$’s parameters
12: for $j \leftarrow 1$ to $T$ do
13: $\hat{G}_{T_0} \leftarrow \nabla_{T_0} \mathcal{L}(f_{T_0}, P_{train})$
14: $T_{\theta'} \leftarrow T_{\theta'} - \lambda_1 \cdot \hat{G}_{T_0}$
15: end for
16: $G_i \leftarrow \nabla_{T_{\theta'}} \mathcal{L}(f_{T_{\theta'}}, P_{test})$
17: end for
18: $M_0 \leftarrow M_0 - \lambda_2 \sum_{i=1}^{K} G_i$, $\triangleright$ outer update
19: end for
20: return $\hat{M}$

Algorithm 2 The $\ell_2$ Norm Attack of MetaSimulator

Input: Input image $x \in \mathbb{R}^D$ where $D$ is image dimension-
ality, true label $y$ of $x$, feed-forward function $f$ of target model, MetaSimulator $M$, attack objective loss $\mathcal{L}(\cdot, \cdot)$.

Parameters: Warm-up iterations $t$, meta-predict interval
$m$, Bandits exploration $\tau$, finite difference probe $\delta$, learning rate $\eta$.

Output: $x_{adv}$ that satisfies $\|x_{adv} - x\|_2 \leq \epsilon$.

1: Initialize the adversarial example $x_{adv} \leftarrow x$
2: Initialize the gradient to be estimated $g \leftarrow 0$
3: Initialize $D \leftarrow \text{deque}(\text{maxlen} = t) \triangleright$ a bounded
double-ended queue with maximum length of $t$, adding
a full $D$ leads it to drop its oldest item automatically
4: while not successful do
5: $\eta \leftarrow N(0, 1 / D I_{N,N})$
6: $q_1 \leftarrow g + \tau u$, $q_2 \leftarrow g - \tau u$
7: $q_1 \leftarrow q_1/\|q_1\|_2$, $q_2 \leftarrow q_2/\|q_2\|_2$
8: if $i \leq t$ or $(i - t) \bmod m = 0$ then
9: $\hat{y}_1 \leftarrow f(x_{adv} + \delta \cdot q_1)$, $\hat{y}_2 \leftarrow f(x_{adv} + \delta \cdot q_2)$
10: append $\{x_{adv} + \delta \cdot q_1, \hat{y}_1, x_{adv} + \delta \cdot q_2, \hat{y}_2\} \rightarrow D$
11: if $i > t$ then
12: Fine-tune $M$ using $D$ \triangleright Fine-tune $M$ every
$m$ iterations after the warm-up phase
13: end if
14: else
15: $\hat{y}_1 \leftarrow M(x_{adv} + \delta \cdot q_1)$, $\hat{y}_2 \leftarrow M(x_{adv} + \delta \cdot q_2)$
16: end if
17: $\Delta_g \leftarrow \mathcal{L}(\hat{y}_1, y) - \mathcal{L}(\hat{y}_2, y)$
18: $g \leftarrow g + \eta \cdot \Delta_g$, $\triangleright$ $\ell_\infty$ norm attack uses:
19: $\hat{g} \leftarrow \frac{\exp(g \cdot \Delta_g) - (1 - \hat{g}) \cdot \exp(-g \cdot \Delta_g)}{\exp(g \cdot \Delta_g) + (1 - \hat{g}) \cdot \exp(-g \cdot \Delta_g)}$
20: $x_{adv} \leftarrow \prod_{(x,\epsilon)} (x_{adv} + \eta \cdot \text{sign}(g))$ for $\ell_\infty$ norm
21: $x_{adv} \leftarrow \text{Clip}(x_{adv}, 0, 1)$
22: end while
23: return $x_{adv}$

3.3. MetaSimulator Learning

Initialization. Algorithm 1 shows the training procedure.
We sample $K$ tasks randomly to form a mini-batch.
The proposed approach is equipped with a double-network
framework which includes a MetaSimulator network $M$ and
a task-specific network $T$. $T$ is cloned from $M$ for the
learning of each task. Given that $T$ and $M$ use the same network,
the former initializes its parameters using the parameters of
the latter at the beginning of learning each mini-batch.

Meta-train. $T$ uses the gradient descent to update its pa-

ditional supervised learning with a knowledge-distillation-fashioned loss, which matches the fine-tuning step of the attack, thereby connecting the training stage and attack stage.

**Meta-test.** After several iterations, \( \mathbb{T} \) converges and computes the meta-gradient \( G_i \) on the meta-test set of the \( i \)th task. \( G_i \) calculated in Algorithm 1’s line 16 is a high order gradient, thus we call it meta-gradient. Then, meta-gradients \( G_1, \ldots, G_K \) of \( K \) tasks are accumulated as \( \sum_{i=1}^{K} G_i \). Because of the same architecture of the two networks, \( \sum_{i=1}^{K} G_i \) can be used to update the parameters of \( \mathcal{M} \) (the outer update step), which allows this network to learn the general simulation capability over all tasks.

**Loss function.** We adopt a knowledge-distillation-fashioned loss to facilitate MetaSimulator to output a similar prediction with the sampled network \( \mathcal{M}_i \), which is used in both the inner and outer step. Given two queries \( Q_{i,1} \) and \( Q_{i,2} \) of the \( i \)th query pair \( Q_i \), where \( i \in \{1, \ldots, n\} \) and \( n \) is the number of query pairs in the meta-train/meta-test set. The logits outputs of MetaSimulator and \( \mathcal{M}_i \) are denoted as \( \hat{p} \) and \( p \), respectively. The loss function is defined as

\[
\mathcal{L}(\hat{p}, p) = \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_{Q_{i,1}} - p_{Q_{i,1}})^2 + \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_{Q_{i,2}} - p_{Q_{i,2}})^2
\]

The two terms utilize the mean square error (MSE) to push the prediction and the pseudo label close.

### 3.4. MetaSimulator Attack

Algorithm 2 shows the procedure of \( \ell_2 \) norm attack of MetaSimulator; the \( \ell_\infty \) norm attack can be easily obtained by modifying line 18 and line 19. To be consistent with the training stage, we use the steps from Bandits attack [25] to generate query pairs. Query pairs of the first \( t \) iterations are fed to the target model (warm-up phase). These queries and corresponding outputs are collected into a double-ended queue \( \mathbb{D} \), which records the historical trajectory of the evolving queries. \( \mathbb{D} \) drops the oldest item once it is full, which is beneficial for focusing on new queries when fine-tuning \( \mathcal{M} \) using \( \mathbb{D} \). After the warm-up phase, subsequent query pairs are fed into the target model every \( m \) iterations, and the fine-tuned MetaSimulator \( \mathcal{M} \) takes the rest. The gradient estimation step follows Bandits because of its leading performance. The attack objective loss function involved in the gradient estimation is shown in Eqn. 4, which is maximized during attack.

\[
\mathcal{L}(\hat{y}, t) = \begin{cases} 
\max_{j \neq t} \hat{y}_j - \hat{y}_t, & \text{if untargeted attack;} \\
\hat{y}_t - \max_{j \neq t} \hat{y}_j, & \text{if targeted attack;}
\end{cases}
\]

\( \hat{y} \) is the logits output of the target model, \( t \) is the target class in the targeted attack and the true class in the untargeted attack, and \( j \) indexes the other classes.

### 3.5. Relation to Prior Works

In this study, meta-learning and black-box attacks are not simply 1+1 combined. The differences are as follows.

**Relation to Meta-learning.** MetaSimulator is designed to simulate the outputs of any target models accurately when the feeding queries are only slightly different from each other. To this end, it is learned from the intermediate data (query sequences & outputs) of attacks in a knowledge-distillation-fashioned way. None of existing meta-learning methods learns a simulator in this way, they all focus on the few-shot classification or reinforcement learning problem.

**Relation to Meta Attack.** Meta Attack [15] shares the similar design philosophy with ours. However, it trains an auto-encoder on benign image & gradient pairs, rather than the data of the real attack. Hence it is weak in targeted attack. Furthermore, the accurate prediction of the gradient map is difficult for its lightweight auto-encoder, which results in its poor performance in the high-resolution map.

**Special Design.** The proposed method connects the training stage and attack stage seamlessly: the partition of meta-train set and meta-test set in each task (Fig. 2) matches two successive groups of attack iterations (Fig. 1); it learns from the outputs of multiple networks in a knowledge-distillation way (via Eqn. 3), which is the same with the fine-tuning step of the attack. In addition, Algorithm 2 feeds queries to \( \mathcal{M} \) and the target model alternately to learn from the outputs of latest queries. A bounded length double-ended queue \( \mathbb{D} \) is used to collect them, which is beneficial for focusing on new queries when fine-tuning \( \mathcal{M} \) using \( \mathbb{D} \) periodically. The periodic fine-tuning is crucial for high attack success rate in a difficult attack (e.g., targeted attack in Fig. 3b).

### 4. Experiment

#### 4.1. Experiment Setting

**Dataset and Target Models.** The experiments are conducted in CIFAR-10 [28], CIFAR-100 [28], and TinyImageNet [48] datasets. Following previous studies [13, 58], 1,000 tested images are randomly selected from their validation sets for evaluation, all of which are correctly classified by WRN-28 [59], ResNet-110 [20] and Inception-v3 [55]. For the selection of the hyperparameters, we select another 1,000 images to build the validation set. In CIFAR-10 and CIFAR-100 datasets, we follow Yan et al. [58] for the selection of the following target models: (1) a 272-layer PyramidNet+Shakedrop model (PyramidNet-272) [19, 57], which is state-of-the-art network on CIFAR-10; (2) a model obtained through a neural architecture search called GDAS [12], which has a different architecture from the networks used in training MetaSimulator; (3) WRN-28 [59] with 28 layers and 10 times width expansion; and (4) WRN-40 [59] with 40 layers and 10 times width expansion. In TinyImageNet dataset, we select ResNeXt-101 (32x4d) [56],
ResNeXt-101 (64x4d), and DenseNet-121 with a growth rate of 32 [21] as target models.

**Compared Methods.** The Bandits attack [25] is selected as the baseline. The compared methods include nature evolution strategy (NES) [24], Bandits [25], Meta Attack [15], random gradient-free (RGF) method [9], and prior-guided RGF (P-RGF) method [9]. We directly use the official implementation code of Meta Attack for the experiments. The training data (i.e., images & gradients) of this attack are generated using the pre-trained networks in the present study. To achieve a fair comparison, we translate the code of NES, RGF, and P-RGF from the official implementation of TensorFlow [1] into the PyTorch [47] version in experiments. The RGF and NES attacks calculate the gradients of the images by using random perturbations. The P-RGF attack improves the query-efficiency of RGF by incorporating the prior gradient of a surrogate model, which adopts ResNet-110 [20] in the CIFAR-10 & CIFAR-100 datasets and ResNet-101 [20] in the TinyImageNet dataset. Given that the official implementations of RGF and P-RGF only support the untargeted attack, both attacks are excluded in the targeted attack experiments. All methods are limited to the maximum of 10,000 queries in the untargeted and targeted attack experiments for all datasets. Following Yan et al. [58], we set the same $\epsilon$ value for all attacks, which is 4.6 in $\ell_2$ norm attack and 8/255 in $\ell_{\infty}$ norm attack. The configurations of all methods are in the appendix.

**Pre-trained Networks.** To evaluate the capability of simulating unknown target models, the selection of $\mathcal{M}_1, \ldots, \mathcal{M}_n$ in Algorithm 1 should be different from the target models. Thirteen networks are selected to generate the training data of CIFAR-10 and CIFAR-100 datasets and fifteen networks for the TinyImageNet dataset (detailed in the appendix). In experiments of the attack of defensive models (Sec. 4.4), we re-train MetaSimulator and MetaAttack by removing all ResNet networks, because all defensive models adopt a ResNet-50 backbone.

**Method Setting.** In the training stage, we generate query sequence data $Q_1, \ldots, Q_{100}$ in each task, the meta-train set $\mathcal{D}_{\text{mt}}$ consists of $Q_1, \ldots, Q_{50}$, and the meta-test set $\mathcal{D}_{\text{mte}}$ consists of $Q_{51}, \ldots, Q_{100}$. We select ResNet-34 [20] as the backbone of the MetaSimulator to achieve the balance between the performance and the training-time efficiency. MetaSimulator is trained for three epochs over 30,000 tasks, where 30 tasks are randomly sampled to form a mini-batch. In the attack stage, the fine-tuning iterations is set to 10 in the first fine-tuning and reduced to 5 for subsequent ones. The default parameters are listed in Tab. 1.

**Evaluation Metric** Following previous related studies [23, 25, 5, 58], we use the attack success rate, the average and the median of queries to evaluate the performance of the attacks. In the untargeted attack, an attack is successful if the prediction of the target model is different from the ground-truth. In the targeted attack, an attack is successful if the example is predicted as the target class.

**4.2. Ablation Study**

The ablation study inspects MetaSimulator using two aspects: validating the benefit of meta training and the effect of key parameters.

**Meta training.** We validate the benefits of meta training by comparing the performance of the proposed attack algorithm with different backbones of the simulator. The

| name | backbone | default | description |
|------|----------|---------|-------------|
| $\lambda_1$ | ResNet-34 | 0.001 | the backbone of MetaSimulator. |
| $\lambda_2$ | ResNet-34 | 0.01 | learning rate for the gradient descent in the inner update. |
| maximum query times | | 10,000 | |
| $\ell_2$ norm attack's $\epsilon$ | | 4.6 | the radius of $\ell_2$ norm ball in $\ell_2$ norm attack. |
| $\ell_{\infty}$ norm attack's $\epsilon$ | | 8/255 | the radius of $\ell_{\infty}$ norm ball in $\ell_{\infty}$ norm attack. |
| $\eta_{\ell_2}$ | | 0.1 | the image learning rate for updating image. |
| $\eta_{\ell_{\infty}}$ | | 1/255 | the image learning rate for updating image. |
| $\eta_{\text{target}}$ | | 0.1 | OCO learning rate for updating the gradient g. |
| $\eta_{\text{g}}$ | | 1.0 | OCO learning rate for updating the gradient g. |
| Bandits hyperparameter for generating $q_1$ and $q_2$. | | 0.3 | the Bandits hyperparameter for generating $q_1$ and $q_2$. |
| $\ell$ | | 0.01 | the finite difference probe for perturbing input image. |
| $t$ | | 0.1 | the finite difference probe for perturbing input image. |
| deque D's length | | 12 | update iterations of learning meta-train set. |
| meta-predict interval m | | 10 | iteration times of fine-tune during attacking. |
| fine-tune iterations | | 5 | target model only takes the queries every m iterations. |
| warm-up iterations t | | 10 | maximum length of D. D drops its oldest item if it is full. |

![Figure 3](image-url)
MetaSimulator $\mathcal{M}$ is replaced with two networks for comparison, i.e., Simulator$_{\text{rand}}$: a randomly initialized ResNet-34 network, Simulator$_{\text{vanilla}}$: a ResNet-34 network that is trained on the same data with MetaSimulator but without using meta-learning. Tab. 2 shows the comparative experimental results of $\ell_2$ norm attack in CIFAR-10 dataset. The results indicate that MetaSimulator achieve the highest attack success rate and the minimum number of queries, thereby confirming the benefit of using meta training. To inspect the effect of meta training on the simulation capacity, we calculate the average MSE between the logits outputs of the simulators and the target model at each attack iteration (Fig. 3a). MetaSimulator achieve lower average MSE than Simulator$_{\text{vanilla}}$ in all attack iterations; the smaller deviation of the prediction of the former than that of the latter signifies its satisfactory simulation capability.

In the control experiments, we check the effect of the key parameters of MetaSimulator on the CIFAR-10 dataset, including meta-predict interval, warm-up iterations, and deque $\mathcal{D}$’s maximum length. Only one parameter is adjusted and the others are fixed as setting in Tab. 1.

**Meta-predict interval** is the iteration interval that uses MetaSimulator $\mathcal{M}$ to make predictions. A larger meta-predict interval results in more queries but the less opportunity of fine-tuning MetaSimulator. Consequently, MetaSimulator cannot accurately simulate the target model in a difficult attack, resulting in a low attack success rate. Fig. 3b justifies our hypothesis that the attack success rate curve of the targeted attack (red curve) declines because this attack is more difficult than the untargeted attack.

**Warm-up** Fig. 3c suggests that more warm-up iterations yield a higher Avg. query, because more queries are fed into the target model in the warm-up phase.

**Deque $\mathcal{D}$’s maximum length** Fig. 3d shows that no obvious trend of the query number is present in the different maximum lengths.

### 4.3. Results of Attacks on Normal Models

The normal model is the classification model without the defensive mechanism. We conduct untargeted attack and targeted attack experiments on the target models described in Sec. 4.1. In the targeted attack, the target class is set to $y_{adv} = (y + 1) \mod C$ for all attacks, where $y_{adv}$ is the target class, $y$ is the true class, and $C$ is the class number. RGF [9] and P-RGF [9] are excluded from the targeted attack experiments because their official implementations do not support the targeted attack. Tab. 3 and Tab. 4 show the results of the CIFAR-10 and CIFAR-100 datasets. Tab. 6 and Tab. 7 show the results of the TinyImageNet dataset. The results reveal that: (1) MetaSimulator can gain up to $2 \times$ reduction in the average and median of the queries compared with the baseline Bandits; and (2) MetaSimulator obtains significantly fewer queries and higher attack success rate than Meta Attack [15]. The poor performance of Meta Attack can be attributed to its high gradient estimation cost (i.e., it uses the ZOO method [7] for gradient estimation). Furthermore, the lightweight auto-encoder of Meta Attack cannot precisely predict the gradient map, especially in high-resolution images (e.g., Tab. 6 and Tab. 7).

Tab. 3, Tab. 4, Tab. 6, and Tab. 7 show the results when the maximum number of queries is set to 10,000. To further inspect the attack success rate at different maximum queries, we conduct untargeted attack experiments by limiting different maximum queries of each adversarial example and comparing their attack success rates. Fig. 4 demonstrates the superiority of the proposed approach in terms of attack success rate. Fig. 5 demonstrates the relation between query and attack success rate from the different angle. It displays the average number of queries that reaches different desired success rates. Specifically, given a desired success rate $\alpha$ and the query list $Q$ of all samples, the average query is defined as:

$$\text{Avg. Query} = \frac{\sum_{i=1}^{N} Q_{i}}{N}, \quad \text{where } Q_{i} = Q[Q \leq P_{\alpha}], \quad (5)$$

where $P_{\alpha}$ is the $\alpha$th percentile value of $Q$ and $N$ is the length of $Q$. Fig. 5 shows that the proposed approach is more query-efficient than other attacks and the gap is amplified for higher success rates.

![Figure 4](image1.png)

**Figure 4:** Comparison of attack success rate at different limited maximum queries in untargeted attack under $\ell_\infty$ norm, where ResNext-101 indicates ResNext-101(32×4d).

### 4.4. Results of Attacks on Defensive Models

Tab. 5 shows the experimental results of attacking defensive models. ComDefend [27] and Feature Distillation [34]...
are equipped with a denoiser to transform the input images before feeding to the target model. PCL [41] introduces a new loss function to maximally separate the intermediate features of each class. All defensive models adopt a ResNet-50 backbone. Results of Tab. 5 conclude:

(1) MetaSimulator exhibits the best performance in breaking ComDefend. Its attack success rate is higher than Bandits, 20.3% higher and 24.3% higher in CIFAR-10 and CIFAR-100 datasets, respectively.

(2) Meta Attack demonstrates poor performance in ComDefend and Feature Distillation because of its unsatisfactory attack success rate. By contrast, the proposed approach can break this type of defensive models with a high success rate.

Figure 5: Comparison of the average query per successful image at different success rates in the untargeted attack under $\ell_\infty$ norm. More results are shown in the appendix.

(a) PyramidNet-272 in CIFAR-100 (b) WRN-28 in CIFAR-100
5. Conclusion

In this study, we present a novel black-box attack method that trains a generalized substitute model named MetaSimulator to mimic unknown target models accurately for reducing the query complexity. To eliminate the requirement of target models in training, the query sequences generated during attacking many different pre-trained networks are used as the training data with the form of multi-tasks. The proposed approach is equipped with a double-network framework with a learning-to-learn strategy to learn the general simulation capability. After training, a large fraction of queries can be transferred to MetaSimulator, thereby significantly reducing the query complexity of the attack.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pages 265–283, 2016.

[2] Moustafa Alzantot, Yash Sharma, Surjyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. Genattack: Practical black-box attacks with gradient-free optimization. In {Proceedings of the Genetic and Evolutionary Computation Conference}, pages 1111–1119, 2019.

[3] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. 2019.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause, editors, {Proceedings of the 35th International Conference on Machine Learning}, volume 80 of {Proceedings of Machine Learning Research}, pages 274–283, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[5] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In {European Conference on Computer Vision}, pages 158–174. Springer, 2018.

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In 2017 {IEEE} Symposium on Security and Privacy (SP), pages 39–57. IEEE, 2017.

[7] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In {Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security}, pages 15–26. ACM, 2017.

[8] Minhao Cheng, Simranjit Singh, Pin-Yu Chen, Sijia Liu, and Dawn Song. Deep神经网络的防攻击方法与改进。In {Advances in Neural Information Processing Systems}, pages 10932–10942, 2019.

[9] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. In {Advances in Neural Information Processing Systems}, pages 10932–10942, 2019.

[10] Jackson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F de Souza, and Thiago Oliveira-Santos. Copycat cmm: Stealing knowledge by persuading confession.
with random non-labeled data. In 2018 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2018. 3

[11] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In 28th USENIX Security Symposium (USENIX Security 19), pages 321–338, Santa Clara, CA, Aug. 2019. USENIX Association. 2, 3

[12] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1761–1770, 2019. 6

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018. 1, 6

[14] Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7714–7722, 2019. 3

[15] Jiawei Du, Hu Zhang, Joey Tianyi Zhou, Yi Yang, and Jiashi Feng. Query-efficient meta attack to deep neural networks. In International Conference on Learning Representations, 2020. 2, 3, 6, 7, 8, 9, 10, 13, 14, 15

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1126–1135. JMLR. org, 2017. 3

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples (2014). arXiv preprint arXiv:1412.6572. 1

[18] Chuan Guo, Jacob R Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q Weinberger. Simple black-box adversarial attacks. arXiv preprint arXiv:1905.07121, 2019. 3

[19] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5927–5935, 2017. 6

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016. 6, 7

[21] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017. 7

[22] Qian Huang, Isay Katsman, Horace He, Zeqi Gu, Serge Belongie, and Ser-Nam Lim. Enhancing adversarial example transferability with an intermediate level attack. In Proceedings of the IEEE International Conference on Computer Vision, pages 4733–4742, 2019. 2, 3

[23] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In Proceedings of the 35th International Conference on Machine Learning, ICML 2018, July 2018. 1, 2, 7, 9, 10, 13, 15

[24] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. arXiv preprint arXiv:1804.08598, 2018. 2, 7

[25] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In International Conference on Learning Representations, 2019. 1, 2, 3, 6, 7, 9, 10, 13, 14, 15

[26] Nathan Inkawhich, Kevin J Liang, Lawrence Carin, and Yiran Chen. Transferable perturbations of deep feature distributions. arXiv preprint arXiv:2004.12519, 2020. 2, 3

[27] Xiaojun Jia, Xingxing Wei, Xiaochun Cao, and Hassan Feroosh. Comdefend: An efficient image compression model to defend adversarial examples. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 6084–6092, 2019. 8, 10, 14, 15, 20

[28] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[30] T. Lee, B. Edwards, I. Molloy, and D. Su. Defending against neural network model stealing attacks using deceptive perturbations. In 2019 IEEE Security and Privacy Workshops (SPW), pages 43–49, May 2019. 2

[31] Yandong Li, Lijun Li, Liqiang Wang, Tong Zhang, and Boqing Gong. Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks. arXiv preprint arXiv:1905.00441, 2019. 3

[32] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. arXiv preprint arXiv:1611.02770, 2016. 3

[33] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In Proceedings of 5th International Conference on Learning Representations, 2017. 2, 3

[34] Zihao Liu, Qi Liu, Tao Liu, Nuo Xu, Xue Lin, Yanzhi Wang, and Wujie Wen. Feature distillation: Dnn-oriented jpeg compression against adversarial examples. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 860–868. IEEE, 2019. 8, 10, 14, 15, 17, 18, 20

[35] Daniel Lowd and Christopher Meek. Adversarial learning. In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pages 641–647, 2005. 2, 3

[36] Chen Ma, Chenxu Zhao, Hailin Shi, Li Chen, Junhai Yong, and Dan Zeng. Metaadvdet: Towards robust detection of evolving adversarial attacks. In Proceedings of the 27th ACM International Conference on Multimedia, MM 19, page 692701, New York, NY, USA, 2019. Association for Computing Machinery. 3

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations, 2018. 1, 3
A. Experiment Settings

A.1. Compared Methods

NES attack: Hyperparameters for NES attack [23] are listed in Tab. 8. In the targeted $\ell_p$ norm attack setting, NES uses a randomly chosen image of target class as the initial image to optimize. Then, it iteratively modifies the image so that its distance from the original input image is getting smaller and smaller. Finally, the $\ell_p$ distance between the produced adversarial image and the original image is within a preset $\epsilon$. Thus, the hyperparameters of NES attack are carefully tuned in untargeted and targeted attack separately, so as to achieve the highest attack success rate. The experiments of NES attack are conducted by using the PyTorch implementation, which is translated from the original TensorFlow implementation.

Meta Attack: Hyperparameters of Meta Attack [15] are listed in Tab. 9, all the default values are specified by the official implementation code. Note that in the experiments of targeted attack and TinyImageNet dataset, the meta interval $m$ is set to 3, and which is set to 5 in the untargeted attack of CIFAR-10/CIFAR-100 dataset. Meta Attack uses the official PyTorch implementation to experiment, and the gradients of training data are generated by using the same pre-trained models with MetaSimulator, as listed in Tab. 13.

RGF and P-RGF attack: Hyperparameters of RGF [9] and P-RGF [9] attacks follow the official implementation. The experiments of RGF and P-RGF are conducted by using the Pytorch implementation that is translated from the official TensorFlow implementation.

Bandits attack: Hyperparameters of Bandits attack [25] are a subset of the hyperparameters of MetaSimulator, and both methods set the same value to ensure the fairness of comparison. The OCO learning rate is used to update the prior, which is an alias of gradient $g$ for updating the input image.

A.2. Pre-trained Networks and Target Models

Pre-trained networks: In the training of MetaSimulator and Meta Attack, we collect various types of networks to generate the training data. In our experiments, 14 types of networks and 15 types of networks are chosen in CIFAR-10/CIFAR-100 and TinyImageNet datasets, respectively. The list of these networks and their training configurations are listed in Tab. 13.

Target models: In order to evaluate the performance of attacking unknown models, we specify the target models to equip with completely different architectures from the pre-trained networks. The target models and their complexity are listed in Tab. 12.

| Dataset       | Attack | Norm | Hyperparameter       | Value |
|---------------|--------|------|----------------------|-------|
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
| CIFAR-10      | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_2$ norm ball | 4.6   |
|               |        |      | $h$, image learning rate | 2.0   |
|               | $\ell_2$ | $\ell_\infty$ | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
|               |        |      | $h$, image learning rate | 1e-2  |

Table 8: The hyperparameters of NES attack [23], where the sampling variance $\sigma$ for gradient estimation is set to $1e-3$, the number of samples per draw is set to 50, and the maximum queries is set to 10,000 for both untargeted and targeted attack.
Table 9: The hyperparameters of Meta Attack [15], where the binary steps is set to 1, and the solver of gradient estimation adopts the adam for all types of attacks in all datasets.

| Dataset  | Attack | Norm | Hyperparameter | Value |
|----------|--------|------|----------------|-------|
| CIFAR-10/100 | Untargeted | $\ell_2$ | $h$, image learning rate | $1\times 10^{-2}$ |
|          |        |      | top-$q$ coordinates for estimating gradient | 125 |
|          |        |      | $m$, meta interval | 5 |
|          |        |      | use_tanh, the change-of-variables method | true |
|          |        |      | $\epsilon$, radius of $\ell_2$ norm ball | 4.6 |
|          | Targeted | $\ell_\infty$ | $h$, image learning rate | $1\times 10^{-2}$ |
|          |        |      | top-$q$ coordinates for estimating gradient | 125 |
|          |        |      | $m$, meta interval | 5 |
|          |        |      | use_tanh, the change-of-variables method | false |
|          |        |      | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |
| TinyImageNet | Untargeted | $\ell_2$ | $h$, image learning rate | $1\times 10^{-2}$ |
|          |        |      | top-$q$ coordinates for estimating gradient | 125 |
|          |        |      | $m$, meta interval | 3 |
|          |        |      | use_tanh, the change-of-variables method | true |
|          |        |      | $\epsilon$, radius of $\ell_2$ norm ball | 4.6 |
|          | Targeted | $\ell_\infty$ | $h$, image learning rate | $1\times 10^{-2}$ |
|          |        |      | top-$q$ coordinates for estimating gradient | 125 |
|          |        |      | $m$, meta interval | 3 |
|          |        |      | use_tanh, the change-of-variables method | false |
|          |        |      | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |

Table 10: The hyperparameters of RGF [9] and P-RGF [9] attacks, where the surrogate model of P-RGF attack adopts ResNet-110 in CIFAR-10 and CIFAR-100 datasets, and it adopts ResNet-101 in TinyImageNet dataset.

| Norm  | Hyperparameter | Value |
|-------|----------------|-------|
| $\ell_2$ | $h$, image learning rate | 2.0 |
|       | $\sigma$, sampling variance | $1\times 10^{-4}$ |
|       | $\epsilon$, radius of $\ell_2$ norm ball | 4.6 |
| $\ell_\infty$ | $h$, image learning rate | 0.005 |
|       | $\sigma$, sampling variance | $1\times 10^{-4}$ |
|       | $\epsilon$, radius of $\ell_\infty$ norm ball | 8/255 |

Table 11: The hyperparameters of Bandits attack [25].

| Dataset | Network | Model Details | Params(M) | MACs(G) | Layers |
|---------|---------|---------------|----------|---------|--------|
| CIFAR-10 | PyramidNet-272 | | 26.21 | 4.55 | 272 |
|          | GDAS | | 3.02 | 0.41 | 20 |
|          | WRN-28 | | 36.48 | 5.25 | 28 |
|          | WRN-40 | | 55.90 | 8.08 | 40 |
| CIFAR-100 | PyramidNet-272 | | 26.29 | 4.55 | 272 |
|          | GDAS | | 3.14 | 0.41 | 20 |
|          | WRN-28 | | 36.54 | 5.25 | 28 |
|          | WRN-40 | | 55.90 | 8.08 | 40 |
| TinyImageNet | DenseNet-121 | | 7.16 | 0.23 | 121 |
|          | ResNeXt-101 | | 42.54 | 0.65 | 101 |
|          | ResNeXt-101 (32 × 4d) | | 81.82 | 1.27 | 101 |

Table 12: The details of black-box target models which are used for evaluating attack methods, where MAC is the multiply-accumulate operation count.

ComDefend [27] and Feature Distillation [34] are based on the input image preprocessing, the attack success rate of Meta Attack is rather low under this type of defense. In contrast, the performance of MetaSimulator is significantly better than that of baseline Bandits [25] in all defensive models, especially the ComDefend and Feature Distillation.

B.2. Figures of Experimental Results

The experimental results are demonstrated in the form of two types of figures. The first type limits different maximum queries of attacks and compare their attack success
Table 13: The list of pre-trained networks and their training configurations, these networks are used to generate the training data of MetaSimulator and Meta Attack. All ResNet networks are excluded in the experiments of attack defensive models.

| Dataset         | Network           | epochs | lr    | lr decay epochs | lr decay rate | weight decay | layers | Hyperparameters | other hyperparameters |
|-----------------|-------------------|--------|-------|-----------------|---------------|--------------|--------|-----------------|------------------------|
| CIFAR-10/100    | AlexNet           | 164    | 0.1   | 81, 122         | 0.1           | 5e-4         | 9      | -               | growth rate:12, compression rate:2 |
|                 | DenseNet-100      | 300    | 0.1   | 150, 225        | 0.1           | 1e-4         | 100    | -               | growth rate:40, compression rate:2 |
|                 | DenseNet-190      | 300    | 0.1   | 150, 225        | 0.1           | 1e-4         | 100    | -               | block name: BasicBlock widening factor:8, cardinality:8 |
|                 | PreResNet-110     | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 110    | -               | block name: BasicBlock |
|                 | ResNeXt-29 (8 × 64d) | 300   | 0.1   | 150, 225        | 0.1           | 5e-4         | 29     | -               | block name: BasicBlock |
|                 | VGG-19 (BN)       | 164    | 0.1   | 81, 122         | 0.1           | 5e-4         | 19     | -               | - |
|                 | ResNet-20         | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 20     | -               | block name: BasicBlock |
|                 | ResNet-32         | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 32     | -               | block name: BasicBlock |
|                 | ResNet-44         | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 44     | -               | block name: BasicBlock |
|                 | ResNet-50         | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 50     | -               | block name: BasicBlock |
|                 | ResNet-56         | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 56     | -               | block name: BasicBlock |
|                 | ResNet-110        | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 110    | -               | block name: BasicBlock |
|                 | ResNet-1202       | 164    | 0.1   | 81, 122         | 0.1           | 1e-4         | 1202   | -               | block name: BasicBlock |
| TinyImageNet    | VGG-11            | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 11     | -               | - |
|                 | VGG-11 (BN)       | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 11     | -               | - |
|                 | VGG-13            | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 13     | -               | - |
|                 | VGG-13 (BN)       | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 13     | -               | - |
|                 | VGG-16            | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 16     | -               | - |
|                 | VGG-16 (BN)       | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 16     | -               | - |
|                 | VGG-19            | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 19     | -               | - |
|                 | VGG-19 (BN)       | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 19     | -               | - |
|                 | ResNet-18         | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 18     | -               | - |
|                 | ResNet-34         | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 34     | -               | - |
|                 | ResNet-50         | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 50     | -               | - |
|                 | ResNet-101        | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 101    | -               | - |
|                 | DenseNet-161      | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 161    | growth rate: 32 | - |
|                 | DenseNet-169      | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 169    | growth rate: 32 | - |
|                 | DenseNet-201      | 300    | 1e-3  | 100, 200        | 0.1           | 1e-4         | 201    | growth rate: 32 | - |

Table 14: Experiment results of untargeted ℓ∞ norm attack on defensive models in TinyImageNet dataset, all defensive models adopt the backbone of ResNet-50. Scatter is Feature Scatter [60], Distillation is Feature Distillation [34].

| Dataset         | Attack            | Attack Success Rate | Avg. Query | Median Query |
|-----------------|-------------------|---------------------|------------|--------------|
| TinyImageNet    | NES [23]          | 74.8%               | 1326       | 2034         | 3225       | 500 | 250 | 650 | 2250 |
|                 | RGF [9]           | 43.7%               | 1855       | 1239         | 1619       | 765 | 408 | 810 | 985  |
|                 | P-RGF [9]         | 48.8%               | 1986       | 1482         | 2231       | 866 | 436 | 810 | 985  |
|                 | Meta Attack [15]  | 9.2%                | 2173       | 2636         | 4187       | 1431 | 1824 | 3381 | 2602 |
|                 | Bandits [25]      | 52.4%               | 785        | 2147         | 1272       | 68 | 206 | 1152 | 182  |
|                 | MetaSimulator     | 54.6%               | 265        | 1070         | 746        | 28 | 148 | 666 | 154  |

The results show that the highest red bars (MetaSimulator) are located in the low query number’s area, thereby confirming that most adversarial examples of MetaSimulator have the fewest queries. The second type of figures measure the average number of queries over successful images by reaching a desired success rate, it demonstrates the relation between query and attack success rate from the different angle. Fig. 8 and Fig. 10 show the results.
Figure 6: Comparison of the attack success rate at different limited maximum queries in TinyImageNet dataset.

Figure 7: Comparison of attack success rate at different limited maximum queries using $\ell_{\infty}$ norm attack in CIFAR-100 dataset, where PyramidNet indicates PyramidNet-272.
Figure 8: Comparison of the average query per successful image at different desired success rates in CIFAR-100 dataset.

Figure 9: Comparison of the attack success rate at different maximum queries on defensive models with ResNet-50 backbone, where Distillation indicates Feature Distillation [34].
Figure 10: Comparison of the average query per successful image at different desired success rates on defensive models with the backbone of ResNet-50, the experiments are conducted by using untargeted $\ell_\infty$ norm attack in CIFAR-10/CIFAR-100 datasets, where Distillation indicates Feature Distillation [34].

Figure 11: The histogram of query number in the CIFAR-10 dataset, where PyramidNet indicates PyramidNet-272.
Figure 12: The histogram of query number in the CIFAR-100 dataset.
Figure 13: The histogram of query number in the TinyImageNet dataset.

Figure 14: The histogram of query number using untargeted attacks on defensive models.