# Publishing Efficient On-device Models Increases Adversarial Vulnerability

***Abstract*—Recent increases in the computational demands of deep neural networks (DNNs) have sparked interest in efficient deep learning mechanisms, *e.g.*, quantization or pruning. These mechanisms enable the construction of a small, efficient version of commercial-scale models with comparable accuracy, accelerating their deployment to resource-constrained devices.**

**In this paper, we study the security considerations of publishing on-device variants of large-scale models. We first show that an adversary can exploit on-device models to make attacking the large model easier. In evaluations across 19 DNNs, by exploiting the published on-device models as a prior, the adversarial vulnerability of the original commercial-scale models increases by up to 100x. We then show that the vulnerability increases as the similarity between a full-scale and its efficient model increase. Based on the insights, we propose a defense, *similarity-unpairing*, that fine-tunes smaller models with the objective of reducing the similarities. We evaluated our defense on all the 19 DNNs and found that it reduces the transferability up to 90% and the number of queries required by a factor of 10–100x. Our results suggest that further research is needed on the security (or even privacy) threats caused by publishing those efficient siblings.**

***Index Terms*—Deep Neural Networks, Efficient Deep Learning, Adversarial Vulnerability**

## I. Introduction

Deep neural networks (DNNs) are vulnerable to *adversarial examples* [1]: an adversary can craft human-imperceptible perturbations to inputs so that models make controlled mistakes. Yet, conducting real-world adversarial attacks is challenging. Many existing machine learning (ML) systems which carry incentives for potential adversaries, *e.g.*, spam filters or abuse detection [2–4], are deployed with server-side models that have limited query access. To fool such systems, an attacker has to conduct black-box adversarial attacks [5–9], which typically requires thousands of queries for crafting a single adversarial example. Also, additional safety mechanisms, *e.g.*, banning malicious users, hinder the adversary's capabilities [10, 11]

A recent trend in ML deployment is to publish DNN models to user devices. Diverse efficient deep learning mechanisms, ranging from compression, *e.g.*, quantization or pruning [12–17], to designing new efficient architectures [18–21], have been proposed to bring commercial-scale server-side models to devices with constrained resources. Pushing on-device models enables service continuity by making predictions offline and reduces the operational cost of service providers. However, this *also* makes it easier for an adversary to reverse-engineer those models and use them to their advantage. Since on-device models are likely constructed by the same providers who trained the server-side counterparts, they likely share many similarities, *e.g.*, they are trained on the same (or very similar) data or use similar architectures (but differ in size).

In this paper, we study the additional increase in vulnerability caused by the common practices of efficient deep learning. Specifically, we ask the questions:

> *Does publishing of on-device models make ML systems vulnerable to adversarial attacks? And if the answer is yes, then what can we do to remedy it?*

We focus on the black-box adversarial attacks as they exploit the technique, *i.e.*, estimating the gradients of the target model from query outputs, applicable to other security or privacy threats, such as model extraction or model inversion [22, 23].

We begin by a comprehensive analysis that characterizes the success of black-box adversarial attacks both with and without access to on-device models. We analyze 19 different pairs of models constructed from six different efficient deep learning mechanisms on two image classification benchmarks. We find that adversarial examples crafted on on-device models transfer with up to 100% success. Moreover, an adversary requires 10–100$\times$ fewer queries to fool the server model and have a higher attack success if they have access to an on-device model.

We next study and characterize this increased vulnerability to understand why it occurs. We find that (i) training on-device models longer reduces the vulnerability, and (ii) the more the difference between server and on-device models' architectures, the less vulnerability is. This observation suggests a *trade-off* between the security and the efforts in constructing on-device models in efficient deep learning techniques. We show more accurate techniques induce a higher vulnerability. We then propose three metrics to measure model similarity, and show that our metrics effectively predict how vulnerable a model will be by having the on-device variant.

Informed by this characterization, we propose a defense that allows training on-device models while minimizing—but not completely eliminating—the increased vulnerability. Our *similarity-unpairing* defense fine-tunes the client model with the objective of making it dis-similar to the server-side model. We comprehensively evaluate the effectiveness of our defense in the network pairs we test in our vulnerability analysis. The adversarial examples crafted our fine-tuned models transfer 50–80% less than the attacks with the original on-device model, and we force the adversary to use use 2–10$\times$ more queries to achieve similar or fewer successes in black-box attacks. While promising, future work will be necessary to completely eliminate this increased vulnerability.

**Contributions.** We summarize our contributions as follows:

- We present a new vulnerability where an attacker exploits on-device variants of server-side models to increase the adversarial vulnerability. We formalize this as a security game and identify the goals for the attacker and defender.
- We show that the attacker can increases the vulnerability of server-side models significantly by exploiting their on-device variants. The adversarial examples crafted on on-device models transfer at most 100% to those models while reducing the query complexity by 10–100×.
- We further characterize the power of using on-device models obtained by six different mechanisms. We find a trade-off between security and the efforts in constructing those on-device priors.
- We study three metrics that allow us to quantify how much two models are *similar* and evaluate their effectiveness in capturing the vulnerability.
- We propose a defense, similarity-unpairing, that forms an objective function for reducing the similarities between on-device and server models. We present three strategies to incorporate this defense into efficient deep learning and demonstrate their effectiveness.

## II. Background

We start by reviewing the paradigm of efficient deep learning and adversarial attacks and relates the two research fields.

### A. *Efficient Deep Learning*

Efficient deep learning develops a wide range of mechanisms for increasing the efficiency of the final trained model while preserving its original accuracy. This is a vast research field; we study the most popular techniques including training new efficient architectures [18, 19, 24, 25] (possibly with neural architecture search [26–29]), knowledge distillation [30–32] to transfer knowledge from a larger model into a smaller model, quantization [12–15, 33] to reduce the size of the model parameters on disk, and pruning [16, 17] to reduce the total number of connections in a neural network. We also study a recent direction, *once-for-all*, where a single model can be trained to support multiple different efficient scales at the same time [21]. In §IV-A, we provide details of these methods.

### B. *(Black-box) Adversarial Attacks*

Adversarial examples [1, 34] allow an adversary to arbitrarily change a neural network's predictions by adding small perturbations to test-time inputs. Many adversarial-example crafting algorithms [34–36, 36–39] have been proposed to benchmark the worst-case sensitivity of a neural network to input perturbations in *white-box* settings, where an adversary has full knowledge of the target neural network. For a given test-time sample $(x, y)$, these attacks search for an input $x'$ that maximizes the loss between a model's output and the true label $\mathcal{L}(f_\theta(x), y)$ while its perturbations are bounded $||x' - x||_{\ell_p}$ to be human-imperceptible.

Our paper focuses on the more realistic *black-box* settings, where an adversary can only access the target model $f_\theta$ through API accesses [6–9, 40–43], and cannot directly compute gradients on the model. Black-box attacks introduce a new constraint: adversaries must minimize the total number of queries made to $f_\theta$. Prior work presented two different strategies: *transfer-based* and *optimization-based* attacks.

**Transfer-based attacks.** Transfer-based attacks exploit the *transferability* phenomenon of adversarial examples [40]: it turns out that if an adversary generates an adversarial example on one model (a "surrogate model"), this exact adversarial example often fool another remote target model. Prior work [41] showed that the effectiveness of transfer-based attacks depends on how *similar* a surrogate is to the target.

**Optimization-based attacks.** If query access to the remote system is possible, early work proposed techniques to first reconstruct a similar model and then using that as a transfer source [9, 44, 45]. However, future optimization-based black-box attacks are often strictly more powerful. They generate adversarial examples by defining an objective function and iteratively perturbing the input to solve that objective.

There are two general attack approaches. Gradient estimation attacks [5, 6, 42, 46, 47] approximate the target model's gradient from query outputs (*i.e.*, softmax confidence scores), and run standard white-box attacks using this estimated gradient. Gradient-free attacks [48–53] only rely on the model's hard decision (*i.e.*, the predicted class), but often require (many) more queries to the target. Our work focuses on gradient estimation attacks as we aim to give the adversary every possible advantage when designing our defense.

**Query attacks with transfer priors.** A final direction unifies the two above attack techniques [6, 7], and exploits transfer priors to perform a query attack. In our experiments we use P-RGF [6] which works by exploiting surrogate models available to the adversary. We use as surrogates an on-device model that has been reverse-engineered by the adversary.

## III. Problem Formulation and Defense Goals

### A. *Threat Model*

Black-box adversarial attacks are a far more practical threat today [54–58], and so we focus on these attacks in the paper. For example, many *abuse detection* tasks, *e.g.*, spam/phishing detection or not-safe-for-work (NSFW) detection, only allow query access to the classifiers running on servers For simplicity, we consider the task of *image classification* because it is the domain that has seen most research on adversarial examples. However, our scenario could also be easily extended to other applications, *e.g.*, text classification. (see Appendix A for the discussion about the practicality of this threat model).

As mentioned in §II-B, black-box attacks can be made much more efficient when transfer priors are available. We consider the specific scenario where *defender* supplies the surrogate to the adversary "for free" by releasing an efficient on-device model that the adversary can use as the transfer prior.

**Notation.** We use the following notation throughout the paper.

- **Adversary (A):** An entity who fools a model by constructing adversarial examples.

- **Model provider (P):** An entity that trains models, and then use them to provide services to general public.
- **Server-side model ($f_{\theta_s}$):** The trained model available remotely on the server.
- **On-device model ($f_{\theta_o}$):** The efficient version of $f_{\theta_s}$ that is released publicly and is on user devices.

The provider **P** offers two types of models:

- **White-box:** The adversary has complete access to the parameters of the model, and can run arbitrary attacks; the on-device models fall into this category.
- **Black-box with limited API access:** The adversary is only allowed to access a query interface and cannot obtain gradients from the model, as happens in the server-side setting. For example, **P** is an image hosting provider and uses a server-side model to detect NSFW images. The model's decision affects whether the user will be further allowed to upload images or not. **A** can only observe the outcome of uploading a few image.

**Adversary objective and cost.** The attacker receives an input $x$, has query access to the server-side model $f_{\theta_s}$, and generates an adversarial example $x'$ so that $f_{\theta_s}$ misclassifies $x'$. In our threat model, **A** has complete white-box access to the on-device model $f_{\theta_o}$ and is allowed to make a limited number of queries to $f_{\theta_s}$. The attack cost is measured as the number of times **A** queries to $f_{\theta_s}$ to craft $x'$. In transfer-based attacks, **A** only uses $f_{\theta_o}$ for crafting; thus, the cost is zero.

## B. Attacker and Defender Security Game

We begin by recalling the standard security game (denoted Game A), used throughout adversarial machine learning that defines the *robust accuracy* of a machine learning model $f_{\theta_s}$.

(1) The defender trains a model $f_{\theta_s}$ with training algorithm $\mathcal{T}_1$. This model can be arbitrarily large and is intended to reach state-of-the-art accuracy at the desired task.
(2) The defender grants the adversary query access to $f_{\theta_s}$.
(3) The adversary samples a benign example $x \leftarrow \mathcal{X}$ from the data distribution, and generates an adversarial example $x'$ on $f_{\theta_s}$ by making queries to this model. The adversary *fails* if $f_{\theta_s}$ correctly classifies $x'$; otherwise, adversary *succeeds* and we measure the *cost* of the attack as the number of times the adversary queries $f_{\theta_s}$.

This security game changes when we focus on the problem of this paper and grant the adversary access to an on-device model that the defender has also constructed. Under this setting, the new security Game B proceeds as follows:

(1) The defender creates two models: $f_{\theta_s}$, using the same $\mathcal{T}_1$, and a new model $f_{\theta_o}$ with an efficient deep learning algorithm $\mathcal{T}_2$. As before, $f_{\theta_s}$ is large and accurate; and this time, the on-device model $f_{\theta_o}$ should be small and efficient for inference, but still achieve high accuracy.
(2) The defender grants the adversary query access to $f_{\theta_s}$ and sends the adversary the on-device model $f_{\theta_o}$.
(3) By exploiting the white-box access to $f_{\theta_o}$, the adversary generates a new adversarial example $x''$. The **success** criteria and the **cost** metric remain unchanged; importantly, only $f_{\theta_s}$ needs to be fooled and only queries to $f_{\theta_s}$ count against the cost.

**Definition 1.** The *vulnerability increase* is defined by the difference in queries necessary to fool the model, formally:

$$Pr[\mathcal{A} \text{ succeeds at Game B}] - Pr[\mathcal{A} \text{ succeeds at Game A}].$$

A more refined analysis can be made by considering the success rate and the reduction in queries necessary to succeed on average. While we generally find these two to be highly correlated, in principle, they need not be, and so we make our definition with respect to success rate and use query count as an additional metric if the success rates are comparable.

**Defender objective.** The defender's objective is to develop a new training algorithm $\mathcal{T}_2'$ that minimizes the vulnerability increase. Note here that $\mathcal{T}_1$ is a fixed function and can not be changed: the server training algorithm is likely a complex setup that is designed to yield the highest quality $f_{\theta_s}$ possible, and it is unacceptable to reduce the quality of the model $f_{\theta_s}$ in order to permit releasing on-device models $f_{\theta_o}$.

Note that this game is only interesting if the server model $f_{\theta_s}$ is not trivially fooled from the beginning. If the adversary, even without access to $f_\text{o}$, can win at the game with probability $\sim 1.0$, then the total vulnerability can not increase significantly by having the transfer prior $f_\text{o}$. However if this happens, then it is true that releasing the model will not cause harm, because the server model is already trivial to evade.

## C. What Makes This Game Tractable?

Over the past decade, there has been limited progress towards "solving" the problem of adversarial examples [36, 59–63]. The best approaches available either significantly harm accuracy [60], cause $100-1000\times$ increase in computations [61], or do not scale to state-of-the-art models [61, 62].

We expect our problem formulation is one that can be reasonably solve because *the defender does not need to train models outright robust*; not to transfer attacks, not to query attacks, and certainly not to gradient-based attacks. Instead, all the defender needs to do is *not to make the situation worse*.

**A trivial, uninteresting, but perfect solution.** There is a straightforward solution to this problem: the defender could just train the on-device model using a completely different training setup, on a completely different training dataset, using a completely different model architecture and hyperparameters. Because this is something the adversary could have done themselves, the advantage at the above game is by definition 0, and this would amount to a perfect solution to our problem.

There are two reasons this solution is undesirable.

- It considerably increases the defender's training complexity. The defender now must maintain two independent training setups, must fix bugs in two independent training algorithms, must gather two different training datasets, and in general must do everything twice.
- The on-device model will be less accurate if trained from scratch than derived from the server model. One of the

main benefits of deriving a small on-device model from a larger pre-trained model is that the resulting on-device model can be made more accurate. Training a smaller model completely from scratch often does not reach the same accuracy levels. While in principle, the defender could entirely re-create the server model training setup and then compress this model down to an on-device model for the sole purpose of obtaining a high accuracy on-device variant, this is an even larger complexity.

**Goals.** The objective of the defender, therefore, is to find a *nontrivial* solution to this problem, and minimize the increased vulnerability of the server model to black-box adversarial examples without having to construct an entirely new and complex training pipeline just to achieve this goal.

## IV. Evaluating the Power of On-Device Priors

We first study this problem from the adversary's perspective and quantify to what extent publishing on-device models increases the vulnerability of the server-side models to black-box adversarial attacks. We outline our experimental setup and methodology (§IV-A) followed by the vulnerability analysis (§IV-B). In §IV-D, we characterize the vulnerability considering the costs a victim requires to spend to obtain on-device models. We further show there is no-free lunch in efficient deep learning (§IV-E): lower-cost techniques to construct on-device models lead to the increase of the vulnerability more than higher-cost techniques. We lastly discuss the security implications of our results and the desiderata for a defense.

### A. Experimental Setup and Methodology

**Datasets.** We run our experiments with two popular image classification benchmarks: CIFAR10 [64] and ImageNet [65]. CIFAR10 is a 10-class classification dataset, with 50k training and 10k testing images of 32×32 pixels. ImageNet contains real-world images of 224×224 pixels with 1000 categories.

**Metrics.** We use two metrics to quantify the vulnerability:

(1) In transfer-based attacks, we design a new metric, relative fooling rate (RFR)[1], to measure the transferability. RFR is a standardized metric that allows us to compare the vulnerability across different datasets, models, and attacks:

$$\mathrm{RFR}(f_{\theta_s}, S, S^o, S^s) = \frac{A(f_{\theta_s}, S) - A(f_{\theta_s}, S^o)}{A(f_{\theta_s}, S) - A(f_{\theta_s}, S^s)}$$

where $A(f_\theta, S)$ denotes the accuracy of a model $f_\theta$ over a set of test-time samples $S$. $S^s$ and $S^o$ are the adversarial examples of $S$ crafted on $f_{\theta_s}$ and $f_{\theta_o}$, respectively. RFR computes how often the adversarial examples $S^o$ can be effective in causing an accuracy drop of $f_{\theta_s}$, compared to the effectiveness of the white-box adversarial examples $S^s$ directly crafted on $f_{\theta_s}$. RFR will be close to 1, when most adversarial examples $S^o$ crafted on $f_{\theta_o}$ transfer to $f_{\theta_s}$; otherwise, RFR will be near 0.

(2) In optimization-based attacks, we measure the number of queries to the target (# Queries) required for crafting adversarial examples on average. We limit the number of queries an adversary can make to 4000 for each sample, following the prior work [8, 9]. If the adversary cannot fool the target after spending 4000 queries, we stop crafting and count it as an attack failure. To quantify the attack success, we also measure the fooling rate (FR). FR is defined as the ratio of adversarial examples crafted by the attacker successfully fool the target.

**Methodology.** We examine the vulnerability of server-side models to black-box attacks when an adversary can exploit on-device models as prior. For transfer-based attacks, we craft adversarial examples using the on-device models and use them to attack the target (server-side) models. We use two canonical adversarial-example crafting algorithms, FGSM and PGD-10. In black-box optimization-based attacks, we use the P-RGF attack formulated by Cheng *et al.* [6]. This attack requires a prior model, and for this we use each of the on-device models. In each cases, we craft adversarial examples on the same 1000 samples randomly chosen from the test data. We limit the perturbations to 8/255 pixels in $\ell_\infty$ norm (a standard value [36]) and fix the step-size to 2/255 pixels in the optimization process. We run this experiment five times.

**On-device Models.** We examine six different mechanisms for constructing on-device models from the target $f_{\theta_s}$:

- **Quantization:** We use 8-bit quantization to compress the target model. Both the parameters and the activations are represented in 8-bit. The on-device model has the size of one-fourth and is computationally efficient as it uses integer-only arithmetic, not floating-point operations.
- **Pruning:** We use $\ell_1$-unstructured pruning, proposed by Li *et al.* [16]. We gradually increase the sparsity by 5% from 0–100% and stop if the accuracy drop becomes more than 4% of the target. Pruned models we create have ∼50% sparsity.
- **Knowledge Distillation (KD):** We construct on-device models with the knowledge distillation proposed by Hinton *et al.* [30]. We set the temperature to 20 and the ratio between the main loss and the distillation loss to 1.0.
- **Neural Architecture Search (NAS):** We further consider on-device models that use the architectures constructed by NAS [20] (*e.g.*, NASNet). This architecture has a smaller number of parameters and achieves compatible accuracy. We train them on the same training data from scratch.
- **Manually-designed Architectures:** There are several architectures manually-designed to reduce the number of parameters while achieving accuracy compatible with the target, *e.g.*, MobileNetV2 or SqueezeNet. We train these models on the same training data from scratch.
- **Once-for-All (OFA) Models.** We also consider the OFA model, proposed by Cai *et al.* [21], where we can extract sub-networks—that have a smaller number of parameters—

[1]RFR has the following advantages over the metrics proposed in prior work: (1) RFR is a standardized metric that enables comparing the effectiveness of different black-box attacks. It is not straightforward with the traditional metrics, *e.g.*, an *accuracy* of $f_{\theta_o}$ over $S'$. (2) We encode the *actual* attack success. Liang *et al.* [66] proposed a metric for quantifying the vulnerability, but we found that the metric won't capture the vulnerability in our scenarios. We include our further analysis in Appendix C.

| Dataset | Original Model ($f_{\theta_s}$) | | On-device Model ($f_{\theta_o}$) | | | | Transfer-based ($\ell_{\text{inf}}$) | | Optimization-based [6] | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Arch. | Acc. (%) | Mechanism | Arch. (New) | Train | Acc. (%) | FGSM | PGD-10 | # Queries | FR (%) |
| CIFAR10 | ResNet50 | 91.0±0.7 | Baseline (as-is) | ResNet50 (✗) | ✗ | 91.0±0.7 | 1.00±0.00 | 1.00±0.00 | 13.7± 1.7 | 100±0.0 |
| | | | Baseline (a new) | ResNet50 (✗) | ✓ | 91.0±0.7 | 0.17±0.05 | 0.66±0.11 | 863.6±314.0 | 82±7.2 |
| | | | Baseline (no $f_{\theta_o}$) | - | - | - | - | - | 3047.2± 27.0 | 28±0.5 |
| | | | Quantization | ResNet50 (✗) | ✗ | 91.0±0.7 | 1.00±0.00 | 1.00±0.00 | 13.9± 2.3 | 100±0.0 |
| | | | Pruning | ResNet50 (✗) | ✗ | 87.9±0.9 | 0.90±0.01 | 1.00±0.00 | 15.1± 4.1 | 100±0.1 |
| | | | Distillation | ResNet18 (✓) | ✓ | 87.9±0.9 | 0.18±0.03 | 0.70±0.04 | 752.3± 86.9 | 85±2.1 |
| | | | Distillation | NASNet (✓) | ✓ | 81.8±1.3 | 0.09±0.02 | 0.40±0.02 | 1713.4± 43.7 | 62±1.2 |
| | | | NAS | NASNet (✓) | ✓ | 84.7±2.8 | 0.11±0.02 | 0.46±0.10 | 1486.1±367.1 | 67±8.7 |
| | | | Manual-arch. | MobileNetV2 (✓) | ✓ | 91.6±0.9 | 0.21±0.03 | 0.72±0.03 | 655.8±118.5 | 87±2.7 |
| | | | Manual-arch. | SqueezeNet (✓) | ✓ | 89.1±0.6 | 0.11±0.02 | 0.45±0.02 | 1507.8±121.5 | 67±3.5 |
| ImageNet | ResNet50 | 75.3±0.5 | Baseline (as-is) | ResNet50 (✗) | ✗ | 75.3±0.5 | 1.00±0.00 | 1.00±0.00 | 12.3± 0.1 | 100±0.0 |
| | | | Baseline (a new) | ResNet50 (✗) | ✓ | 75.3±0.5 | 0.34±0.04 | 0.74±0.10 | 649.9±266.1 | 87±6.2 |
| | | | Baseline (no $f_{\theta_o}$) | - | - | - | - | - | 2558.9± 26.8 | 38±0.4 |
| | | | Quantization | ResNet50 (✗) | ✗ | 75.3±0.5 | 1.00±0.00 | 1.00±0.00 | 12.2± 0.1 | 100±0.0 |
| | | | Pruning | ResNet50 (✗) | ✗ | 73.0±0.0 | 0.44±0.29 | 0.77±0.14 | 600.7±364.3 | 88±7.8 |
| | | | NAS | NASNet (✓) | ✓ | 71.1±1.9 | 0.15±0.02 | 0.31±0.05 | 1952.4±127.3 | 54±3.3 |
| | | | Manual-arch. | MobileNetV2 (✓) | ✓ | 70.8±0.4 | 0.16±0.02 | 0.35±0.03 | 1840.8± 66.5 | 57±1.9 |
| | | | Manual-arch. | SqueezeNet (✓) | ✓ | 50.2±3.8 | 0.08±0.03 | 0.28±0.10 | 1914.3±429.6 | 55±8.0 |
| | [†]OFA (4, 6, 7) | 78.2 | Baseline (as-is) | OFA (4, 6, 7) (✗) | ✗ | 78.2 | 1.00 | 1.00 | 12.9 | 100 |
| | | | Baseline (a new) | MobileNetV2 (✓) | ✓ | 72.0 | 0.27 | 0.53 | 1388.4 | 67 |
| | | | Baseline (no $f_{\theta_o}$) | - | - | - | - | - | 2726.4 | 34 |
| | | | Sub-networks | OFA (4, 6, 3) (✗) | ✗ | 77.0 | 0.87 | 1.00 | 21.5 | 100 |
| | | | | OFA (4, 3, 3) (✗) | ✗ | 75.0 | 0.76 | 0.99 | 40.2 | 100 |
| | | | | OFA (2, 6, 7) (✗) | ✗ | 75.0 | 0.94 | 1.00 | 27.9 | 100 |
| | | | | OFA (2, 6, 3) (✗) | ✗ | 74.0 | 0.80 | 0.99 | 42.3 | 100 |
| | | | | OFA (2, 3, 7) (✗) | ✗ | 71.8 | 0.80 | 0.99 | 38.1 | 100 |
| | | | | OFA (2, 3, 3) (✗) | ✗ | 70.6 | 0.65 | 0.95 | 122.3 | 98 |

[†]Note that we use the single pre-trained model that the original work [21] provides.

TABLE I: **Vulnerability to black-box attacks when on-device models are used as priors,** evaluating on CIFAR10 and ImageNet. We consider six different mechanisms for producing on-device models, and we note the on-device architectures and whether or not the technique requires additional training. Attacks are evaluated with FGSM, PGD-10, and P-RGF. We measure the relative fooling rate (RFR) for FGSM and PGD-10, and for P-RGF, we measure the # Queries and the fooling rate (FR).

without training. One can choose a sub-network from the pre-trained OFA based on the device's resource constraints. We describe the experimental setup in detail in Appendix B.

## *B. Vulnerability Increases When Using On-Device Models*

We show our results in Table I. The first three rows for each server model correspond to baselines: *Baseline (as-is)* directly uses the target as the transfer source, and so is easiest to attack: the transfer rate is 100% because the models are identical, and on average just 13 queries are required to fool the model. Note that P-RGF uses the batch-size of ten for querying. *Baseline (a new)* serves as our perfect-but-uninteresting method described earlier: the adversary trains a completely new model from scratch (using the same architecture); this results in much worse transferability and 60× higher query count than Baseline (as-is). *Baseline (no $f_{\theta_o}$)* does not use any on-device model for the black-box attacks, and thus performs the worst. The remaining rows then exploit various on-device models constructed by using different efficient mechanisms.

**Quantization and pruning increase vulnerability.** To begin with, we study the effect of models constructed by quantization or pruning. In CIFAR10, the transfer-based attacks achieve 0.9–1.0 RFR in both FGSM and PGD, *i.e.*, most adversarial examples crafted on on-device models successfully transfer to the target. And for optimization-based attacks, the adversarial examples achieve 100% attack success rate *by making fewer than 15 queries on average*. In contrast, without any prior, the attacker needs to query the target model 2559–3047 times on average, and even with 300× as many queries, their resulting adversarial examples have a success rate of just 28–38%. This is consistent with prior work [6, 8, 9, 67], which has generally found that that optimization-based attacks require thousands of queries to craft a single adversarial example. However, our results show that just by having the quantized or pruned models as an on-device prior, the attacker reduces the query complexity by two orders of magnitude.

**Once-for-all paradigm also increases the vulnerability.** We also observe that when the attacker has access to the sub-networks obtained from the OFA model, the vulnerability to black-box adversarial attacks increases significantly. The last nine rows in Table I presents the analysis for the OFA models. The hyper-parameters for these models are denoted by a triple $(i, j, k)$ indicating the number of layers in each unit, the expansion ratio for each layer, and the kernel size (respectively). We use the OFA (4, 6, 3) model as the target and choose six different sub-networks with smaller sizes as on-device models. The vulnerability increased by these sub-networks is similar to (or even more) that caused by quantization and pruning.

In transfer-based attacks, we observe the RFR of 0.65–0.94 in FGSM and 0.95–1.00 in PGD-10, which is significantly higher than the baseline (a new) where we see 0.27 and 0.53 RFR in FGSM and PGD-10, respectively. In optimization-based attacks, the attack requires $22\times$–$127\times$ fewer queries than the baseline with no prior and $11\times$–$64\times$ fewer than the baseline that exploits an unrelated MobileNetV2. The attacker also achieves $\sim$100% FR.

**Architectures different from the target are less effective.** We find that the advantage of using an on-device model decreases if the model and the target have different architectures. Using a different architecture require training an on-device model from scratch; thus, we compare our results with Baseline (a new). In CIFAR10, using NASNet or SqueezeNet as an on-device model achieves lower RFRs (0.11 and 0.45 for FGSM and PGD-10, respectively) in transfer-based attacks and $2\times$ higher query complexity in optimization-based attacks than the baseline. ResNet-like architectures, *e.g.*, ResNet18 or MobileNetV2, when used as transfer priors, are more effective than NASNet and SqueezeNet. Using MobileNetV2 on-device models achieves almost the same effectiveness as the baseline. In ImageNet, we observe similar results.

We also observe that knowledge distillation often increases vulnerability even when the two architectures are different. In CIFAR10, a ResNet18 trained by knowledge distillation increases RFRs, requires fewer queries, and has a higher success rate. The vulnerability eventually becomes similar to the baseline, where we use a ResNet50 on-device model trained from scratch. However, when we use NASNet, knowledge distillation does not increase the vulnerability further. We hypothesize that this is because knowledge distillation forces the outputs of a teacher and a student to be similar during training. We will analyze this interaction further in §V-A.

## C. Exploitation in the Real World

In §IV-B, we find that an attacker can increase the risk of black-box adversarial attacks by exploiting on-device priors. We now show obtaining these priors is possible in practice.

Cloud providers offer many services where users can upload their datasets and train on-device models, *e.g.*, Amazon SageMaker Edge or Google's AutoML Edge. Those services are an attractive option for users with limited expertise in machine learning as they reduce the users' effort in optimizing their models for each device. They automatically train high-quality models within the resource limits. In the meantime, service providers utilize efficient deep learning methods to minimize the cost of training on-device models. If a user deploys them to edge, the attacker can reverse-engineer and exploit them for crafting black-box adversarial examples.

**Case study: SageMaker.** We first attempted our attack on Amazon SageMaker Edge, a service that provides both a larger server-side model along with an efficient edge model. Here, we found that the on device models released by SageMaker Edge are *functionally identical* to the large server-side models—they just use different client-side libraries for deep learning computations that are optimized differently. Therefore, this is equivalent to our *as-is* setting. And so in this case, we observe that both the transfer- and optimization-based attacks show $\sim$100% success rate with less than 20 queries. This result, while at the technical level is unsurprising because models are identical, demonstrates the potential pitfalls of releasing on-device models using off-the-shelf tooling.

| $f_s$ | On-device $f_c$ | Acc. ($f_c$) | Transfer- FGSM | PGD10 | Optimization- # Q | FR |
|---|---|---|---|---|---|---|
| **AutoML Vision** | Baseline (no $f$) | 92% | - | - | 3179.6 | 23% |
| | ResNet50 (P) | 91% | 0.05 | 0.12 | 2832.3 | 33% |
| | ResNet50 (Q) | 88% | 0.03 | 0.09 | 2879.3 | 30% |
| | ResNet18 | 89% | 0.03 | 0.10 | 2775.4 | 34% |
| | NASNet | 89% | 0.05 | 0.13 | 2844.1 | 33% |
| | MobileNetV2 | 93% | 0.04 | 0.07 | 2957.8 | 29% |
| | SqueezeNet | 89% | 0.04 | 0.12 | 2731.9 | 35% |
| | AutoML Edge[†] | 93% | 0.14 | 0.10 | 3254.4 | 20% |

[*] We adapt our black-box attacks for this scenario.

TABLE II: **Exploitation of the vulnerability in the real-world.** We first construct two models (server $f_s$ and on-device models $f_c$) using Google AutoML Vision, and we exploit the smaller, on-device model in attacking the cloud-side model.

**Case study: AutoML.** To better understand how these attacks apply in practice, we next use Google's AutoML to construct a server-side model and an on-device model on CIFAR10. In Google's AutoML, a user only has query access to the cloud model, *i.e.*, the predicted class is only visible. One can download the on-device model in TFLite [68] format. Unfortunately, TFLite does not allow to compute the gradients in its format, *i.e.*, the attacker cannot directly apply FGSM or PGD on this model. Instead, we assume that the attacker can reverse-engineer the model parameters from the format and use them to construct a surrogate $f_{o'}$, where we can approximate gradients. We note that this adapted attack is much weaker than directly running the white-box attacks. It enables us to access the vulnerability in real-world scenarios.

Even with this weak, adapted adversary, Table II shows that an adversary can cause a similar or a better adversarial vulnerability than the baseline (no $f_{\theta_o}$). In FGSM, the vulnerability can increase more than twice ($\sim$0.4$\rightarrow$0.14) if the attacker uses the approximated on-device model $f_{o'}$. However, we also find that the vulnerability remains similar in PGD-10, and our adaptive optimization-based attack is ineffective. In most PGD-10 cases, the accuracy drop caused by adversarial examples is low across the board. The main reason is that the cloud models only allow using images in the integer format to query. As we typically compute adversarial perturbations in a floating-point format, rounding them to integers can remove those small changes. Under these constraints, the PGD-10 may already achieve the limit of the accuracy drop.

## D. Characterize the Vulnerability: Cost-Security Analysis

Our analysis showed that efficient deep learning mechanisms demand the training of an on-device model and utilize

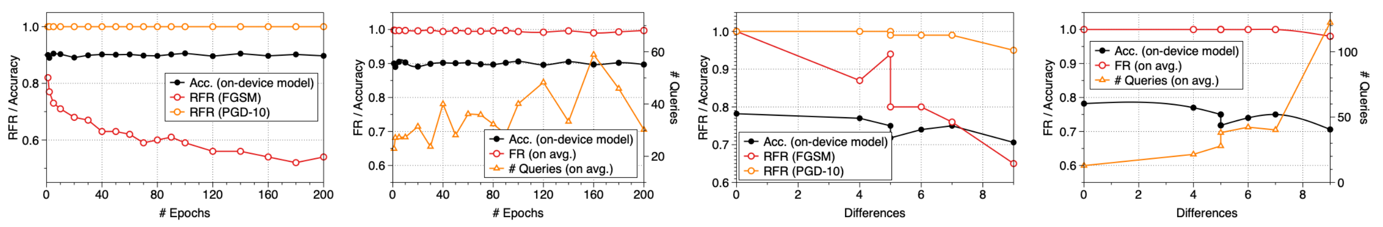

Fig. 1: **Vulnerability decreases as costs of constructing an on-device model increases.** We consider two different costs: the number of training iterations (# Epochs) and the architectural difference between the target and the on-device model. In the left two figures, we fine-tune ResNet50 and measure the variability in the vulnerability. In the right two figures, based on the architectural difference we define between OFA networks, we measure how the vulnerability changes. As the costs increase, RFR decreases in transfer-based attacks, and the query efficiency and the success rate of the optimization-based attack decrease.

different architectures, resulting in less effective priors. These two factors, *i.e.*, the training iterations and the architectural differences, reflect how much effort we require to construct on-device models. We conduct further analysis to characterize the interaction between the vulnerability to black-box adversarial attacks and the costs of building on-device models.

**Longer training iterations reduces the vulnerability.** To validate this hypothesis, we exploit fine-tuned models in transfer-based or optimization-based attacks. We run fine-tuning of a ResNet50 model, pre-trained on CIFAR10, for 200 training iterations (epochs). During training, we store the intermediate models in every 10 epochs. We then use these models in crafting adversarial examples and measure the vulnerability of the original ResNet50 models to black-box adversarial attacks.

The left two figures in Figure 1 show our results. We first observe that the fine-tuning does not reduce the accuracy of the resulting models. We also show that fine-tuning is effective in reducing the vulnerability of a weak transfer-based attack. In the leftmost figure, the RFR of FGSM decreases (0.82→0.54) as the number of training epochs increases (0→200). However, fine-tuning is rendered ineffective when an adversary uses a stronger attack (PGD-10); it shows 1.00 RFR consistently. The second figure shows that the query complexity of the optimization-based attacks increases from 23 to 59 on average, meaning that more training iterations increase the attacker's cost, while the FR stays the same.

**Larger architectural differences reduce the vulnerability.** We validate our hypothesis using the OFA sub-network from Table I. It is challenging to compare the impact of the architectural difference because (i) models have different parameter values that can impact the vulnerability, and (ii) we have no metric to encode architectural differences. We address the first by using those sub-networks—they share the same parameters derived from the OFA model. Thus, we can minimize the impact of parameter differences. We then define the architectural differences between the sub-networks by computing the $\ell_1$ distance between their configurations. For example, the difference between the OFA (4, 6, 7) and OFA (2, 3, 7) models is $\|(4-2)+(6-3)+(7-7)\|_{\ell_1} = 5$. Note that this definition is not an ideal metric to measure the architectural differences—we just use it as a proxy to characterize the interaction between the architectural difference and vulnerability.

We illustrate our results in Figure 1. Overall, we observe that the vulnerability decreases as the architectural difference increases, while the accuracy of the networks remains almost the same. In transfer-based attacks, the RFR decreases from 1.00→0.65 (FGSM) and from 1.00→0.95 (PGD-10), respectively. Like our previous analysis, choosing an architecture different from the target is more effective against a weak (FGSM) attack than the strong (PGD-10) attack. In P-RGF, we find that the query complexity increases from 13→122 on average, while the FR stays similar (100%→98%).

## E. Security Implications

Our analysis results show a trade-off in efficient deep learning. In particular, we look at the interaction between the security and the efforts in constructing on-device models. If we put more effort into deriving on-device models (in terms of the training iterations and designing architectures), the vulnerability decreases. Otherwise, if we use cheaper mechanisms like quantization and pruning, the resulting on-device models increase the vulnerablility of the server-side model.

**Desiderata.** We summarize three conditions we would like out of an efficient deep learning training mechanism.

(1) *No accuracy drop.* A mechanism will be ideal if the resulting on-device models have a compatible accuracy.
(2) *Minimize the training cost.* Mechanisms that do not require training on-device models from scratch are better.
(3) *Reduce the vulnerability.* The vulnerability caused by on-device models should be less than the baselines in Table I.

## V. Similarity Unpairing Reduces Vulnerability

We now study the problem from the eyes of the defender, having shown in the prior section that without applying defensive techniques, there is a trade-off between the computational demands of efficient deep learning mechanisms and the security risk of a target model's to black-box adversarial attacks. If the defender choose more efficient techniques (*e.g.*, pruning, quantizing, or the "once-for-all" strategy) for constructing efficient deep learning models, it will increase the vulnerability to the black-box attacks. However, mechanisms that train models from scratch do not increase vulnerability

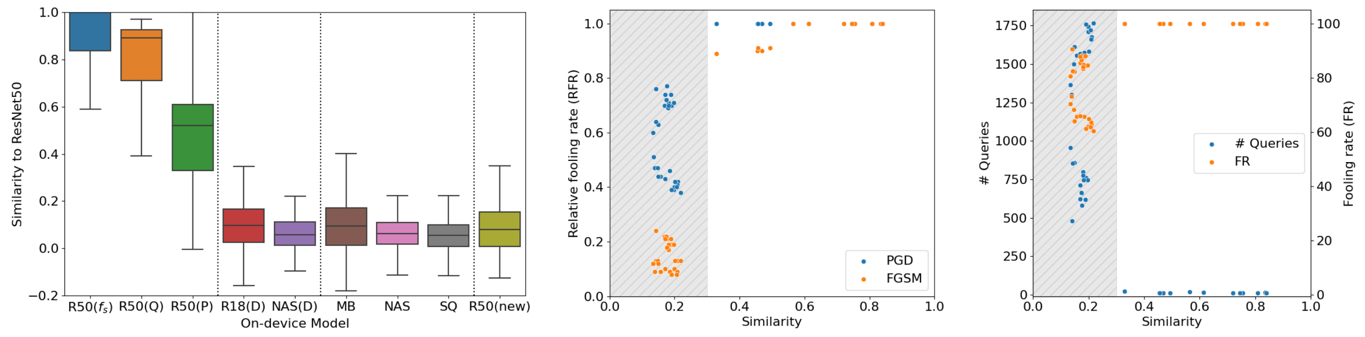

Fig. 2: **Our similarity metric accurately predicts the vulnerability to black-box adversarial examples.** We measure the similarities between the target (ResNet50) and the on-device models from Table I at the input-gradients (left) on CIFAR-10. We plot the interaction between the vulnerability to black-box adversarial attacks and the similarity in the right two figures.

(because the adversary could have done it them self) but are computationally expensive and are also less accurate.

Here, we answer the question: can we reduce the vulnerability to black-box attacks further in an efficient way? We propose a straightforward defense, *similarity-unpairing*, that can actually achieve this by fine-tuning a model to reduce the similarities between $f_{\theta_s}$ and $f_{\theta_o}$. To this end, we first explore metrics that quantify the similarity between the two models. We measure how well the similarity metrics capture the vulnerability to black-box adversarial attacks. We then develop a novel objective function that decreases the similarity between the two models $f_{\theta_s}$ and $f_{\theta_o}$ by fine-tuning the on-device model $f_{\theta_o}$. We finally evaluate the effectiveness of our defense in reducing the vulnerability to black-box adversarial attacks when the attacker exploits the fine-tuned model $f_{o'}$.

## A. Similarity Metrics to Quantify the Vulnerability

We develop similarity metrics to quantify to what extent models produce similar classifications in the hope that they will allow us to train models to be different as a defense. We identify two locations in a model to measure similarity: activations and outputs (*i.e.* logits). We additionally compute the gradients at the input space for the same test-time sample to capture the similarity of the two models' loss surfaces. Using the three metrics (activations, outputs, and input-gradients), we measure the similarities between models in Table I.

**Metrics.** We utilize the cosine similarity loss $C_s$ as a metric to quantify how much outputs obtained from each of the three locations are similar. The loss value will be 1 when they are similar; otherwise, -1. We compute the loss at each location as follows (see the computation details in Appendix D):

(1) *Output:* We compute the loss between the logits.
(2) *Input-gradients:* We compute the loss between the input-gradients (*i.e.*, the gradients computed on the same input with respect to two different models).
(3) *Activations:* We compute the similarity between the activations of a model before the classification head (*i.e.*, the latent representations), commonly regarded as features.

**Results.** Figure 2 shows our results in CIFAR10. We compute the similarities between the target ResNet50 (R50) and the seven different models we construct in Table I. We use the same 1,000 samples used for crafting adversarial examples.

The leftmost figure show the distribution of similarities observed at the input-gradients. We find that the input-gradients reflects the vulnerability better than the two other metrics, *i.e.*, logits and activations (see Appendix E). Quantization and pruning lead to on-device models with the highest similarities. Perhaps unsurprisingly, we find that the similarity decreases as we use the mechanisms that require training a new model from scratch: ResNet18 (D), NASNet (D), MobileNetV2, NASNet, SqueezeNet, and ResNet50 (a new), confirming our observation in §IV-D. We further observe that, compared to the baseline where we traine a new ResNet50 from scratch, the architectural differences decrease the similarity further.

In the right two figures, we plot the interaction between the vulnerability to black-box adversarial attacks and the similarity measured in the input-gradients. We compute them with the models we examine previously in Table I. We first show that the input-gradient similarity directly predicts how easy it is to perform a black-box attack: the model pair that is most similar is the best transfer source, and the least similar model pair is the worst transfer source. And in query attacks (the right figure), as similarity increases, the query efficiency and the FR also increase. We also find that decreasing the input-gradient similarity below 0.4 reduces the vulnerability significantly. In transfer-based attacks, compared to the cases with a similarity score over 0.3, the RFR of FGSM and PGD-10 decrease to 0.1–0.25 and 0.3–0.8 (respectively). In optimization-based attacks, the FR decreases to 55–90%, and the number of queries required increases to 1000–1600. This connection gives the insight to develop techniques that reduce the vulnerability of black-box adversarial attacks in constructing models with efficient deep learning algorithms.

## B. Similarity Unpairing Objective

We present our defense: *similarity unpairing*. The intuition for this defense is straightforward. As we have seen, while we would like to directly convert the model $f_{\theta_s}$ into an efficient

| Objective | Penalize Output-level Sim. | | | | | Penalize Input-level Sim. | | | | | Penalize Feature-level Sim. | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | | Transfer- | | Optimization- | | | Transfer- | | Optimization- | | | Transfer- | | Optimization- | |
| | Acc. | FGSM | PGD10 | # Q | FR | Acc. | FGSM | PGD10 | # Q | FR | Acc. | FGSM | PGD10 | # Q | FR |
| 0.0 | 91% | 0.84 | 1.00 | 46.2 | 99% | 92% | 0.83 | 1.00 | 29.1 | 100% | 92% | 0.83 | 1.00 | 30.1 | 100% |
| 0.001 | 91% | 0.84 | 1.00 | 26.8 | 100% | 91% | 0.85 | 1.00 | 26.1 | 100% | 92% | 0.83 | 1.00 | 30.1 | 100% |
| 0.01 | 91% | 0.83 | 1.00 | 22.6 | 100% | 91% | 0.83 | 1.00 | 40.3 | 100% | 92% | 0.83 | 1.00 | 36.3 | 100% |
| 0.1 | 91% | **0.62** | **0.61** | 819.1 | 82% | 91% | 0.76 | 1.00 | 135.3 | 100% | 91% | 0.83 | 1.00 | 26.2 | 100% |
| 1.0 | 90% | 0.66 | 0.67 | 664.4 | 86% | 91% | **0.08** | **0.09** | **3411.9** | **15%** | 91% | 0.86 | 1.00 | 17.4 | 100% |
| 10.0 | 90% | 0.64 | 0.62 | **841.7** | **80%** | 72% | 0.22 | 0.86 | 191.9 | 97% | 91% | 0.85 | 1.00 | 24.1 | 100% |

TABLE III: **Effectiveness of our similarity unpairing defense.** We measure the vulnerability of $f_{\theta_s}$ to black-box adversarial examples when the attacker uses the model $f_{\theta_{c'}}$ fine-tuned with our similarity-unpairing loss. We use the ResNet50 model $f_{\theta_s}$ trained on CIFAR10. The cases where the attacker is the least successful are highlighted in bold.

model $f_{\theta_o}$, this causes potential harm because an adversary can make use of the input-gradient similarity between $f_{\theta_o}$ and $f_{\theta_s}$. So to prevent this, we fine-tune a model $f_\theta$ and convert it into a new model $f_{\theta'}$ that has similar accuracy, but a different input-gradient landscape. Formally, to reduce the similarity, we fine-tune the model on the following objective function:

$$\mathcal{L}_{ours} = \mathcal{L}_{\text{xe}}\big(f'(x), y\big) + \sum_{i \in S} \lambda_i \cdot \mathbb{E}_{\mathcal{D}}\left[{C_s}^i\big(x, y, f_\theta, f_{\theta'}\big)\right]$$

where $\mathcal{L}_{\text{xe}}$ is the cross-entropy loss, ${C_s}^i$ is the $i$-th similarity function we define in §V-A, $f_\theta$ and $f_{\theta'}$ are the original and fine-tuned models, and $\lambda_i$ is the hyper-parameter controlling the two loss terms. We fine tune for 20 epochs (*i.e.*, just 10% of the total steps to train the server-side model), using the same set of hyper-parameters we use for training. We consider only one similarity ${C_s}^i$ at a time, but we combine them together later on for reducing the vulnerability further. We also train $f_{\theta'}$ from scratch, but this does not offer better protection. We include those additional results in Appendix F.

**Single objective at a time.** Table III shows the effectiveness of penalizing each similarity individually in reducing the vulnerability to black-box adversarial examples. We fine-tune the ResNet50 model on CIFAR10 and then evaluate as before, using the fine-tuned model to craft adversarial examples on the target model, setting $\lambda_i$ in 0.001–10.0 to control the impact of our unpairing loss. In each case, we measure the accuracy of the fine-tuned model $f_{\theta_{c'}}$ and the vulnerability metrics we define. The baseline $\lambda_i = 0$ is included as the first row.

We find penalizing the outputs and the input-gradients effectively reduces the vulnerability to black-box adversarial examples. If we decrease the output-level similarity, the RFRs of the transfer-based attacks reduce from 0.84→0.62 (FGSM) and 1.00→0.61. Additionally, the query complexity increases from 46.2 (baseline) to 841.7, and the success rate decreases from 100% to 80%. Penalizing the similarity of the input-gradients is more effective than reducing the output-level similarity. Here, in the transfer-based attacks, the RFRs are decreased to below 0.09 for both FGSM and PGD-10, with the query attack complexity increasing from 29.1 to 3411.9, and the FR reduces from 100% to 15%. In both cases, we further preserve the accuracy of the fine-tuned models compared to the baselines. However, we find that penalizing the activation-level similarity is not effective in reducing the vulnerability.

| Objectives | | Penalize Multiple Sim. | | | | |
|---|---|---|---|---|---|---|
| Output. | Input. | | Transfer-based | | Optimization-based | |
| $\lambda_1$ | $\lambda_2$ | Acc. | FGSM | PGD10 | # Q | FR |
| 0.01 | 0.01 | 91% | 0.25 | 0.87 | 291.1 | 95% |
| 0.01 | 0.1 | 91% | 0.23 | 0.83 | 344.2 | 96% |
| 0.01 | 1.0 | 91% | 0.05 | 0.08 | 2922.7 | 32% |
| 0.1 | 0.01 | 92% | 0.24 | 0.79 | 381.2 | 94% |
| 0.1 | 0.1 | 92% | 0.20 | 0.77 | 491.5 | 91% |
| 0.1 | 1.0 | 91% | 0.06 | 0.12 | 2647.9 | 39% |
| 1.0 | 0.01 | 91% | 0.18 | 0.64 | 564.4 | 91% |
| 1.0 | 0.1 | 91% | 0.18 | 0.65 | 567.5 | 91% |
| **1.0** | **1.0** | **93%** | **0.06** | **0.08** | **2835.5** | **34%** |

TABLE IV: **Combining multiple unpairing objectives.** We measure the vulnerability when we fine-tune on-device models while decreasing both the outputs and the input-gradients similarity. We highlight the least vulnerable case in bold.

**Combining multiple unpairing objectives.** We further examine if combining multiple objectives simultaneously can reduce the vulnerability more. We fine-tune the ResNet50 model while reducing the similarities at the outputs and the input-gradients. The hyper-parameters $\lambda_1$ and $\lambda_2$ are the weights for the output similarity loss and input-gradient similarity loss. We vary them in 0.01–1.0. (We exclude the activation-level similarity as it is ineffective.)

Table IV shows these results. Combining multiple objectives reduces the vulnerability only slightly: if we set both $\lambda$s to 1.0, the RFRs of FGSM and PGD-10 become 0.06 and 0.08, just $\sim$0.02 less than penalizing only the similarity between input-gradients. Therefore, for the remainder of our experiments, we consider just one loss penalty.

## VI. Safe and Efficient Deep Learning with Similarity Unpairing

In §V-B, we show that our similarity unpairing would reduce the vulnerability of server-side models if the adversary had direct access to the fine-tuned model. However, these fine-tuned models are still large (server-scale) models that can not be used on devices directly. Here, we combine our defense with efficient deep learning mechanisms to construct efficient on-device models with reduced vulnerability. We compare three strategies to construct efficient and safe on-device models:

(1) *Finetune first:* We first fine-tune the model $f_{\theta_s}$ and apply efficient deep learning mechanisms, *e.g.*, the victim fine-tunes a pre-trained model and then applies quantization.
(2) *Finetune last:* We can also apply efficient deep learning mechanisms first to construct the on-device model $f_{\theta_o}$ and then fine-tune it to reduce the vulnerability, *e.g.*, the victim constructs a quantized version of $f_{\theta_s}$ and then fine-tunes the resulting model $f_{\theta_o}$ with our objective function.
(3) *Jointly finetune:* We can further develop novel efficient deep learning mechanisms that return on-device models with reduced vulnerability by incorporating our similarity-unpairing objective into their algorithms, *e.g.*, we add our objective to the distillation loss.

| **Mechanism** | **Finetune first** | **Finetune last** | **Jointly finetune** |
|---|---|---|---|
| Quantization | ✓ | ✓ | ✓ |
| Pruning | ✓ | ✓ | ✓ |
| Distillation | ✓ | ✓ | ✓ |
| New-arch. | ✗ | ✓ | ✗ |
| OFA | ✓ | ✓ | ✗ |

TABLE V: **Applicability of similarity-unpairing.** We show the applicability of our defense to six efficient deep learning mechanisms that our work examines. ✓ indicates that we can take the strategy, while ✗ means that it is not applicable.

Table V shows the applicability of these three strategies to the efficient deep learning mechanisms that we examine. The *finetune-first* (FF) strategy is applicable when a mechanism uses a pre-trained model. If the victim constructs on-device models by designing new architectures, *e.g.*, NAS, or manually-designed architectures, this strategy is not applicable. The *finetune-last* (FL) strategy is always applicable as one can fine-tune pre-trained on-device models with our defense objective. We can apply the *jointly-finetune* (JL) strategy when an efficient deep learning mechanism involves an optimization process. The victim can add our similarity-unpairing loss as an additional objective for the process.

## A. Strawman Solution: Robust Server-side Models

Before we test whether our similarity unpairing reduces the vulnerability, we examine a strawman solution that first constructs an adversarially-robust server-side model and reduces its size so that on-device models are somewhat robust.

**Methodology.** We first run adversarial training (AT) [36]—a standard approach to train adversarially-robust models—on ResNet50. We run AT with PGD-7 that has the perturbation bound of 8/255 pixels in $\ell_\infty$-norm and the step-size of 2/255 pixels. We then utilize the six mechanisms we previously used to construct on-device models in §IV-D. Using the on-device models we construct as transfer priors, we perform black-box adversarial attacks on the robust ResNet50.

**Results.** Due to the space limit, we summarize our results in Appendix H and I. As we reviewed in §III, robust models suffer from the utility loss–their accuracy is 10–20% less than the undefended models. In transfer-based attacks, we observe that robust server-side models cannot reduce the vulnerability increase. Quantization and pruning lead to the RFR of 0.8–1.0. In the NASNet, MobileNetV2, and SqueezeNet cases, we observe the RFR further increases to 0.5–0.6; they were 0.1–0.2 in our results with non-robust server-side models. Nevertheless, we find that robust server-side models reduce the vulnerability increase in query-based attacks. Even in the most vulnerable cases, *i.e.*, baseline (as-is), quantization, and pruning, the FR is 53–64% and, the number of queries needed for crafting a successful adversarial example is 1476–1886. In remaining cases, the FR is significantly reduced to $\sim$25%, and the attacker requires $\sim$3000 queries to construct an adversarial example. It is therefore necessary to construct defenses that allow the release of safe on-device models until the trade-off of training truly robust models is deemed acceptable.

## B. Similarity Unpairing with Quantization

**Methodology.** To apply our defense to the FF setting, we fine-tune a ResNet50 $f_{\theta_s}$ with our similarity-unpairing loss for 10–30 epochs and then quantize the model. In the FL setting, we first quantize the pre-trained ResNet50 in 8-bit and then fine-tune for 10–30 epochs. For finetuning (JL) we develop a quantization-aware training (QAT) scheme by minimizing:

$$\mathcal{L}(f_{\theta_s}, x, y) = \mathcal{L}_{\text{xe}}(f_{\theta_s}(x), y) \ + \alpha \cdot \mathcal{L}_{ours}(f_{\theta_o}(x), y)$$

where $f_{\theta_o}$ is the 8-bit representation of $f_{\theta_s}$, and $\alpha$ is the hyper-parameter between the two loss-terms. We set $\alpha$ to 1.0.

| **Dataset** | **Strategy** | **Acc. (8-bit)** | **Transfer-** | | **Optimization-** | |
|---|---|---|---|---|---|---|
| | | | **FGSM** | **PGD10** | **# Q** | **FR** |
| **CIFAR10** | Baseline | 91% | 1.00 | 1.00 | 13.7 | 100% |
| | FF | 68% | 0.38 | 0.58 | 901.8 | 80% |
| | **FL** | 91% | **0.13** | **0.31** | **1131.9** | **74%** |
| **ImageNet** | Basline | 76% | 1.00 | 1.00 | 13 | 100% |
| | FF | 75% | 0.51 | 0.93 | 140.7 | 99% |
| | **FL** | 73% | **0.43** | **0.91** | **240.00** | **97%** |

TABLE VI: **Effectiveness of similarity-unparing in quantization.** We show the vulnerability of $f_s$ to black-box adversarial examples when we create quantized on-device models $f_{c'}$ by using three strategies. We highlighted the most effective defense strategy and the vulnerability it causes in bold.

**Results.** Table VI shows our results. The baseline is the case where we quantize the pre-trained ResNet50 as-is. We first find that FL is the best strategy at reducing the vulnerability to black-box adversarial examples. In both CIFAR10 and ImageNet, the RFR of the transfer-based attack is reduced to 0.13–0.43 (FGSM) and 0.31–0.91 (PGD-10), compared to the baseline of 1.00. Our defense increases the query complexity from 12–14 to 138–1132 and decreases the FR from 100% up to 74%. However, we also find that FF is not an effective strategy for reducing vulnerability. While it reduces the RFR in the transfer-based attacks, and the query efficiency and the

FR in the optimization-based attack, the defense entails a large accuracy drop (91→68%). In JF, we observe that our QAT results in a model that shows a significant accuracy drop when quantized (91→47% in CIFAR10). We do not measure the vulnerability that $f_{c'}$ causes as our QAT leads to inaccurate models. We analyze the reason for this failure. We find that making a single model with significantly different gradients on the same inputs in its 32-bit and 8-bit representations is extremely difficult while achieving a high accuracy.

## *C. Similarity Unpairing with Pruning*

**Methodology.** Pruning with FF is simple: we fine-tune the ResNet50 with our unpairing loss and then increase sparsity until the accuracy begins to drop, as we did before. In FL, we first prune the pre-trained ResNet50 with 50% sparsity and then fine-tune this model for 10–30 epochs with our unpairing objective. Joint finetuning is again most difficult; we attempted to apply Variational Dropout [69] before the classification head of a network and use it a learned mask that removes neurons from the network. During fine-tuning, we optimize only the dropout layer to minimize our similarity-unpairing objective. Unfortunately, joint finetuning in this way does not work and reduces model accuracy to 20%.

| Dataset | Strategy | Acc. (Pruned) | Transfer- | | Optimization- | |
|---|---|---|---|---|---|---|
| | | | FGSM | PGD10 | # Q | FR |
| CIFAR10 | Baseline | 88% (0.5) | 0.89 | 1.00 | 19.5 | 100% |
| | FF | 89% (0.3) | 0.37 | 0.55 | 743.6 | 83% |
| | **FL** | 91% (0.5) | **0.31** | **0.46** | **880.6** | **81%** |
| ImageNet | Basline | 73% | 0.97 | 1.00 | 12.2 | 100% |
| | **FF** | 73% (0.4) | **0.51** | **0.94** | **148.18** | **99%** |
| | FL | 74% (0.5) | 0.58 | 0.99 | 64.0 | 98% |

TABLE VII: **Effectiveness of similarity-unparing in pruning.** We show the vulnerability of $f_s$ to black-box adversarial examples when we use pruning to construct on-device models $f_{c'}$. The number in parenthesis next to each accuracy is the sparsity. We highlighted the most effective strategy in bold.

**Results.** Table VII shows our results. Pruning the pre-trained ResNet50 is our baselines. Overall, the RFR of the transfer-based attacks is reduced to 0.31–0.51 (FGSM) and up to 0.46 (PGD-10) from 1.00. In CIFAR10, our defense significantly increases the query complexity from 12–19 to 50–881 and decreases the FR from 100% to 81%. We decrease the vulnerability similar to the case of retraining a model from scratch with 85–95% *fewer* training epochs. In ImageNet, we reduce the RFR from 0.97→0.51 (FGSM) and 1.00→0.94 (PGD-10) while increasing query complexity by 10×. We note that this does not mean our defense, on ImageNet models, is less effective than retraining a model from scratch (a new $f_\theta$). Our further analysis in Appendix G implies that one could reduce the vulnerability more by increasing the learning rate or training with longer epochs. In FF, we need to stop pruning at a smaller sparsity (0.3–0.4). Pruning more than 30–40% of the parameters leads to an accuracy drop of more than 20%. Unfortunately, in JF, our proposed pruning makes the accuracy of a fine-tuned model to ∼20%.

## *D. Similarity Unpairing with Distillation*

**Methodology.** We further examine the three strategies to achieve similarity-unpairing during the distillation process. We can first fine-tune the teacher with our unpairing objective and use the fine-tuned teacher for distillation (FF). Here, we hypothesize that the dissimilarity we encode to the teacher can be transferred to the student. We can also fine-tune the student model for a few epochs after the distillation, *i.e.*, we directly reduce the vulnerability. Moreover, we can use our objective function as a regularizer for the distillation loss (JF). In this case, we only consider the input-gradient similarity as minimizing the output similarity can disturb the distillation process, *i.e.*, the distillation does not work if we make the outputs of both the teacher and student dissimilar.

| Network (Student) | Strategy | Acc. (Student) | Transfer- | | Optimization- | |
|---|---|---|---|---|---|---|
| | | | FGSM | PGD10 | # Q | FR |
| ResNet18 | Baseline | 88% | 0.18 | 0.70 | 752.3 | 85% |
| | FF | 90% | 0.19 | 0.73 | 413.5 | 94% |
| | **FL** | 84% | **0.12** | **0.37** | **2337.1** | **43%** |
| | JF | 89% | 0.22 | 0.85 | 352.6 | 94% |
| NASNet | Baseline | 82% | 0.09 | 0.40 | 1713.7 | 62% |
| | FF | 87% | 0.11 | 0.52 | 1052.4 | 79% |
| | **FL** | 88% | **0.09** | **0.41** | **1114.2** | **78%** |
| | JF | 89% | 0.15 | 0.69 | 722.1 | 86% |

TABLE VIII: **Effectiveness of similarity-unparing in distillation.** We analyze the effectiveness of our defense in distillation. We use two students ResNet18 and NASNet in CIFAR10. We highlighted the most effective cases in bold.

**Results.** Table VIII shows our results. Our objective function can reduce vulnerability in all three cases. We run our experiments in CIFAR10 and consider two student networks, ResNet18 and NASNet. We first observe that both FL and JF achieve the least vulnerability to black-box adversarial examples. Compared to the baseline, the RFR reduces by ∼0.10 in FGSM and 0.3–0.4 in PGD-10. The query complexity increases by 2–7×, and the FR is reduced by 10–50%. We also find that FF is more effective in distillation than quantization or pruning. Distillation makes the outputs of a student resemble those of a teacher—the fine-tuned model in our case. Our analysis further shows that one can jointly optimize both the distillation and similarity-unpairing objectives. Unlike the JF strategy used in quantization and pruning, this can reduce the vulnerability while preserving the accuracy of a student.

We additionally compare our similarity-unpairing with the adversarially-robust distillation mechanism proposed by Goldblum *et al.* [70]. We take the ResNet50 model as a teacher and perform this distillation process onto ResNet18 (a student). The student achieves an accuracy of 72%, which is 16% less than the models constructed by our defense mechanism. In the transfer-based attacks, we observe the RFR of 0.04 and

0.26 in FGSM and PGD-10, respectively. The optimization-based attacks achieve the FR of 44% with 2398.3 queries per adversarial-example crafting on average. The robust distillation offers a similar amount of vulnerability decrease, but it sacrifices the accuracy of the resulting models by 10–15%.

## E. Similarity Unpairing with New Architectures

**Methodology.** Here, we examine the scenarios where a victim chooses a compact architecture, designed by architecture search mechanisms (*e.g.*, NASNet) or manually (*e.g.*, MobileNet or SqueezeNet). In this case, we consider the FL or JF strategies as the victim needs to train a network from scratch. We wonder if the defender can push the vulnerability reduction further than just training a new architecture from scratch. In FL, we take a pre-trained model and fine-tune it with our similarity-unpairing objective for 40 more epochs. In JF, we train a network from scratch with our defense objective using the same hyper-parameter for training the pre-trained models.

| Dataset | Net-arch. | Acc. | Transfer- |  | Optimization- |  |
|---|---|---|---|---|---|---|
|  |  |  | FGSM | PGD10 | # Q | FR |
| CIFAR10 | MobileNetV2 | 93% | 0.24 | 0.76 | 580.3 | 89% |
|  |  | 92% | 0.12 | 0.35 | 1173.4 | 75% |
|  | SqueezeNet | 89% | 0.09 | 0.47 | 1547.1 | 67% |
|  |  | 87% | 0.09 | 0.43 | 1345.2 | 73% |

TABLE IX: **Effectiveness of similarity-unpairing in using different architectures.** We analyze the effectiveness of our defense in fine-tuning on-device models with smaller architectures (*i.e.*, MobileNetV2 and SqueezeNet). We only consider the finetune-last strategy. For each architecture, the first row is the baseline, and the second row is our results.

**Results.** Table IX shows our results. We run our experiments with MobileNet and SqueezeNet in CIFAR10. We only report the results from FL as we find JF training has a 20–30% accuracy drop. In FL, we find that our defense reduces the vulnerability to transfer-based attacks by 50% at most. Compared to the baseline where we use undefended MobileNetV2, the fine-tuned model shows 0.12–0.36 less in RFR. In the optimization-based attack, the query efficiency and the FR are reduced by 2$\times$ and 14%. In SqueezeNet, both the undefended and fine-tuned models show a similar vulnerability. SqeezeNet already achieves low vulnerability even if the attacker uses the undefended model; thus, we marginally improves the security.

## F. Similarity Unpairing with Once-for-All

**Methodology.** We finally examine our unpairing's effectiveness when used with the once-for-all (OFA) paradigm [21]. We only consider the FL strategy. FF is not compatible with this paradigm as it derives multiple sub-networks from a pre-trained model—the vulnerability between the pre-trained model and sub-networks will remain the same. We examine JL, but the computations require to optimize our objective while we run the progressive shrinking algorithm is intractable. We consider the OFANet-267—the most vulnerable case—and fine-tune it for 20 epochs with our defense.

**Results.** We find that the fine-tuning reduces the vulnerability of the server model (OFANet-467) to black-box adversarial examples while maintaining a 74% accuracy. In the transfer-based attacks, we achieve a RFR of 0.65 and 0.98 in FGSM and PGD-10, compared to the baseline of 0.94 and 1.00. In the optimization-based attack, the query complexity increases from 28 to 220 on average, but the FR remains almost the same (100%$\rightarrow$97%). We achieve this improvement even when the sub-networks are only fine-tuned for a few epochs.

## VII. Conclusion

This paper introduces a new security consideration when training machine learning models that will be hosted both server-side but also published on-device: the release of the on-device model should not increase the server-side vulnerability to adversarial examples. We have shown that naive release methods, as studied in efficient deep learning, do significantly increase server-side vulnerability, but that by fine-tuning a efficient model before the release with our *similarity-unpairing*, it is possible to reduce the advantage an adversary would have from using this on-device model significantly. We make two categories of conclusions:

**Next steps for researchers.** We have posed a new research problem, and while we believe to have constructed a robust defense, no defense can be perfect. It is an open question if there is a *new* way to design stronger black-box attacks that reduce the effectiveness of similarity-unpairing.

We have answered one question in this direction, but other questions remain. For simplicity, we have studied the case where the defender releases just an efficient model; in practice, however, model providers often update their model over time to improve accuracy. It is unknown whether releasing multiple models with small parameter differences can increase adversarial vulnerability even further or if this practice results in a break of our defense without stronger attacks.

We have shown a trade-off between the costs used for constructing on-device models vs. security. The question still remains if this trade-off is inherent to the problem or if there is a way to design around it.

**Next steps for practitioners.** The security threat we expose is immediately practical in any setting where both server- and on-device models are constructed. If security is a potential consideration, this vulnerability will need to be considered when evaluating the vulnerability of the server-side model.

One immediate consequence of this threat is that any server model where corresponding efficient models have *already* been released should be considered insecure. Future on-device models should only be released if either security is not a consideration or if steps have been taken to reduce the vulnerability. Fortunately, we have found with our defense that even a small amount of fine-tuning can mitigate adversaries from exploiting the on-device model to launch adversarial attacks on the server model more successfully.

We believe that answering those open questions will bring the two seemingly distance objectives closer: improving the security of server-side models against black-box attacks and releasing computationally-efficient on-device models.

## References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6199

[2] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter." *LEET*, vol. 8, no. 1, p. 9, 2008.

[3] "NSFW — CVisionLab," https://www.cvisionlab.com/cases/nsfw, accessed: 2022-08-20.

[4] "Your SafeSearch Setting," https://www.google.com/safesearch, accessed: 2022-08-25.

[5] C.-C. Tu, P. Ting, P.-Y. Chen, S. Liu, H. Zhang, J. Yi, C.-J. Hsieh, and S.-M. Cheng, "Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks," 2020.

[6] S. Cheng, Y. Dong, T. Pang, H. Su, and J. Zhu, "Improving black-box adversarial attacks with a transfer-based prior," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/32508f53f24c46f685870a075eaaa29c-Paper.pdf

[7] A. Ilyas, L. Engstrom, and A. Madry, "Prior convictions: Black-box adversarial attacks with bandits and priors," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=BkMiWhR5K7

[8] F. Suya, J. Chi, D. Evans, and Y. Tian, "Hybrid batch attacks: Finding black-box adversarial examples with limited queries," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1327–1344. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/suya

[9] Y. He, G. Meng, K. Chen, X. Hu, and J. He, "DRMI: A dataset reduction technology based on mutual information for black-box attacks," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1901–1918. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/he-yingzhe

[10] S. Chen, N. Carlini, and D. Wagner, "Stateful detection of black-box adversarial attacks," in *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, ser. SPAI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 30–39. [Online]. Available: https://doi.org/10.1145/3385003.3410925

[11] H. Li, S. Shan, E. Wenger, J. Zhang, H. Zheng, and B. Y. Zhao, "Blacklight: Scalable defense for neural networks against Query-Based Black-Box attacks," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2117–2134. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/li-huiying

[12] K. Asanovic and N. Morgan, "Experimental determination of precision requirements for back-propagation training of artificial neural networks," in *IN PROCEEDINGS 2ND INTERNATIONAL CONFERENCE ON MICROELECTRONICS FOR NEURAL NETWORKS*, 1991, pp. 9–15.

[13] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018. [Online]. Available: https://openreview.net/forum?id=By5ugjyCb

[14] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/3e15cc11f979ed25912dff5b0669f2cd-Paper.pdf

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016. [Online]. Available: http://arxiv.org/abs/1603.05279

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016. [Online]. Available: http://arxiv.org/abs/1608.08710

[17] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 6682–6691. [Online]. Available: https://proceedings.mlr.press/v119/malach20a.html

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[19] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[20] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image

recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[21] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://arxiv.org/pdf/1908.09791.pdf

[22] F. Tramér, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 601–618. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer

[23] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[26] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HylVB3AqYm

[27] H. Cai, J. Lin, Y. Lin, Z. Liu, K. Wang, T. Wang, L. Zhu, and S. Han, "Automl for architecting efficient and specialized neural networks," *IEEE Micro*, vol. 40, no. 1, pp. 75–82, 2020.

[28] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *CoRR*, vol. abs/1611.01578, 2016. [Online]. Available: http://arxiv.org/abs/1611.01578

[29] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *CoRR*, vol. abs/1611.02167, 2016. [Online]. Available: http://arxiv.org/abs/1611.02167

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

[31] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *CoRR*, vol. abs/1612.03928, 2016. [Online]. Available: http://arxiv.org/abs/1612.03928

[32] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[33] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *European Conference on Computer Vision (ECCV)*, 2018.

[34] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," *Lecture Notes in Computer Science*, p. 387–402, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40994-3_25

[35] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[36] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb

[37] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015.

[38] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2016, pp. 2574–2582. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.282

[39] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57.

[40] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," 2016.

[41] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," 2017.

[42] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, *ZOO: Zeroth Order Optimization Based Black-Box Attacks to Deep Neural Networks without Training Substitute Models*. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–26. [Online]. Available: https://doi.org/10.1145/3128572.3140448

[43] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=SyZI0GWCZ

[44] P. Li, J. Yi, and L. Zhang, "Query-efficient black-box attack by active learning," 2018.

[45] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 506–519. [Online]. Available: https://doi.org/10.1145/3052973.3053009

[46] A. N. Bhagoji, W. He, B. Li, and D. Song, “Practical black-box attacks on deep neural networks using efficient query mechanisms,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[47] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, “Black-box adversarial attacks with limited queries and information,” in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 2142–2151. [Online]. Available: http://proceedings.mlr.press/v80/ilyas18a.html

[48] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh, and M. B. Srivastava, “Genattack: Practical black-box attacks with gradient-free optimization,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1111–1119. [Online]. Available: https://doi.org/10.1145/3321707.3321749

[49] N. Narodytska and S. Kasiviswanathan, “Simple black-box adversarial attacks on deep neural networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 1310–1318.

[50] A. Al-Dujaili and U.-M. O’Reilly, “Sign bits are all you need for black-box attacks,” in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SygW0TEFwH

[51] Y. Li, L. Li, L. Wang, T. Zhang, and B. Gong, “Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks,” 2019.

[52] S. Moon, G. An, and H. O. Song, “Parsimonious black-box adversarial attacks via efficient combinatorial optimization,” in *International Conference on Machine Learning (ICML)*, 2019.

[53] J. Chen, M. I. Jordan, and M. J. Wainwright, “Hopskipjumpattack: A query-efficient decision-based attack,” in *2020 ieee symposium on security and privacy (sp)*. IEEE, 2020, pp. 1277–1294.

[54] Y. Dong, H. Su, B. Wu, Z. Li, W. Liu, T. Zhang, and J. Zhu, “Efficient decision-based black-box adversarial attacks on face recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[55] K. Yuan, D. Tang, X. Liao, X. Wang, X. Feng, Y. Chen, M. Sun, H. Lu, and K. Zhang, “Stealthy porn: Understanding real-world adversarial images for illicit online promotion,” in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 952–966.

[56] H. Yu, K. Yang, T. Zhang, Y.-Y. Tsai, T.-Y. Ho, and Y. Jin, “Cloudleak: Large-scale deep learning models stealing through adversarial examples.” in *NDSS*, 2020.

[57] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun, “Physically realizable adversarial examples for lidar object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[58] S. Hussain, P. Neekhara, M. Jere, F. Koushanfar, and J. McAuley, “Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 3348–3357.

[59] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” *arXiv preprint arXiv:1704.01155*, 2017.

[60] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, “Theoretically principled trade-off between robustness and accuracy,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 7472–7482.

[61] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, “Certified robustness to adversarial examples with differential privacy,” 2019.

[62] J. Cohen, E. Rosenfeld, and Z. Kolter, “Certified adversarial robustness via randomized smoothing,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1310–1320. [Online]. Available: https://proceedings.mlr.press/v97/cohen19c.html

[63] F. Tramèr, “Detecting adversarial examples is (nearly) as hard as classifying them,” 2021.

[64] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Tech. Rep., 2009.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[66] K. Liang, J. Y. Zhang, B. Wang, Z. Yang, S. Koyejo, and B. Li, “Uncovering the connections between adversarial transferability and knowledge transferability,” in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 6577–6587. [Online]. Available: https://proceedings.mlr.press/v139/liang21b.html

[67] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, “Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks,” in *28th USENIX*

Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 321–338. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/demontis

[68] "TensorFlow Lite — ML for Mobile and Edge Devices," https://www.tensorflow.org/lite, accessed: 2021-12-02.

[69] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2498–2507.

[70] M. Goldblum, L. Fowl, S. Feizi, and T. Goldstein, "Adversarially robust distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3996–4003.

[71] M. Guo, Y. Yang, R. Xu, Z. Liu, and D. Lin, "When nas meets robustness: In search of robust architectures against adversarial attacks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 631–640.

## Appendix A
## Practicality of the Threat Model

We discuss practical scenarios where an adversary exploits on-device models to attack original, server-side models.

**A server-side model used for multiple services.** We consider settings where the victim trains a (server-side) model and uses it as a building block for multiple services the victim offers. For instance, Google can train a classifier detecting NSFW photos and uses it (1) to filter out the images uploaded to Google Drive and (2) to detect YouTube videos with sensitive content. They can also train a model that acts as a profanity filter and uses it (1) to filter out spam emails and (2) to remove websites containing bad language from their search results.

**Evading these filters is challenging.** As the models (*i.e.*, filters) are typically deployed to the servers, where an adversary does not have white-box access, they harness black-box attacks to evade the filtering. However, black-box attacks, such as transfer-based or query-based attacks, are less successful or require a high query complexity than the white-box attacks. The victim can also deploy security mechanisms, such as rate-limiting, user authentication, or returning only hard labels, that increases the costs and the complexity of black-box attacks.

**Adversaries can exploit on-device models for evasion.** To overcome this challenge, the attacker can use on-device models the victim releases. Suppose that Google pushes a NSFW filter to mobile devices to reduce the networking bandwidth and protect user privacy. As the filter will be available to anyone has a mobile device, the attacker reverse-engineer the filter and use it to generate adversarial examples and test their attack. Since there is no limit on what the adversary can do with the device, it is fair to assume the model will be on the attacker's hand. Once the model is there, the attacker can use it for generating YouTube video that evades Google's filtering mechanisms. It will be the same for the profanity filter case.

## Appendix B
## Experimental Setup in Detail

**Setup.** We implemented our analysis framework using Python v3.8 and PyTorch v1.8.0[2] that supports CUDA 11.3 for accelerating computations by GPUs. We run all our experiments on a computing cluster, where we have a machine equipped with 2 Intel Xeon 6248R 3.00GHz 48-core processors, 512GB of RAM, and 8 Nvidia Tesla V100 and 8 Nvidia A40 GPUs.

**Hyperparameters.** We train our CIFAR10 network for 200 epochs from scratch using the same set of hyper-parameters the original studies used. In ImageNet, we use pre-trained models offered by PyTorch libraries[3]. In the case of training ImageNet models from scratch, we use the same hyper-parameters used for constructing pre-trained models[4].

## Appendix C
## Metrics for Quantifying the Vulnerability

**Setup.** We evaluate the effectiveness of the metrics, widely-used in the field of adversarial examples, in capturing the adversarial vulnerability in our setting. We examine two popular metrics: (i) the accuracy drop of a model—typically used in the prior work on adversarial examples, and (ii) the metrics proposed in a recent work [66] on transferability of adversarial examples. We use them to quantify the vulnerability in Table I.

| **Server $f_s$** | | **On-device $f_o$** | | | **Metrics in [66]** | | **Acc. drop (on $f_s$ and $f_o$)** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Arch.** | **Acc.** | **Mech.** | **Arch** | **Acc.** | **FGSM** | **PGD10** | **FGSM** | | **PGD10** | |
| **R50** | 92% | as-is | R50 | 92% | 1.00 | 1.00 | 61% | 61% | 92% | 92% |
| | | new $f_c$ | R50 | 91% | 0.28 | 0.22 | 74% | 16% | 91% | 79% |
| | | Quant | R50 | 92% | 1.00 | 1.00 | 60% | 60% | 92% | 92% |
| | | Pruning | R50 | 91% | 0.97 | 0.97 | 61% | 58% | 91% | 91% |
| | | Distill | R18 | 89% | 0.26 | 0.18 | 59% | 12% | 89% | 70% |
| | | Manual | MV2 | 93% | 0.27 | 0.16 | 68% | 16% | 93% | 67% |
| | | Manual | SN | 89% | 0.12 | 0.09 | 59% | 4% | 89% | 40% |

TABLE X: **Comparing the metrics proposed by prior work.** We evaluate the two vulnerability metrics offered in literature: the accuracy drop caused by adversarial examples and the metrics proposed by [66] in CIFAR10. We compute the metrics between the pair of server and on-device models we examined in Table I. We use adversarial examples crafted by FGSM and PGD-10 on the on-device models.

**Results.** Table X shows our results. R50, R18, MV2, and SN refer to ResNet50, ResNet18, MobileNetV2, and SqueezeNet architectures, respectively. In each column from the left, where we measure the accuracy drop, we show the accuracy degradation of on-device $f_{\theta_o}$ and server $f_{\theta_s}$ models caused by adversarial examples crafted on $f_{\theta_o}$. We first observe that within each attack case, the metric captures the vulnerability somehow, *i.e.*, if the vulnerability decreases, the metrics are reduced as well. However, we also find that those metrics are sometimes misleading. For example, the metrics [66] are smaller in PGD-10 (0.18) than in FGSM (0.26), while PGD-10 is a much stronger attack. In distillation, the accuracy drop

[2]https://pytorch.org/

[3]https://pytorch.org/vision/stable/models.html

[4]https://github.com/pytorch/vision/tree/main/references/classification

of the server model caused by PGD-10 (70%) is much larger than that of FGSM (12%).

While the accuracy drop (or the fooling rate) captures the strength of an attack well, we observe that it cannot capture how many adversarial examples successful on $f_{\theta_o}$ can also fool the attack target ($f_{\theta_s}$). For example, the adversary would not use a set of adversarial examples, unsuccessful on $f_{\theta_o}$—the model used to craft them—for attacking the server-side model. To address this problem, we propose a new metric, *relative fooling rate* (RFR), that can capture how much the attacker can be successful relatively more/less than the baseline. In our work, we set the baseline as the white-box attacks on $f_{\theta_s}$ and compare them with the black-box attacks with $f_{\theta_o}$.

## APPENDIX D
## COMPUTING SIMILARITY METRICS

In §V-A, we compute our similarity metrics as follows:

- **Output** similarity defined as:

$$C_s{}^{\mathrm{o}} = \mathrm{mean}_{\mathcal{D}_\mathrm{s}}\{cosine(f_{\theta_s}(x), f_{\theta_o}(x))\}$$

  where $f_{\theta_s}$ and $f_{\theta_o}$ are server-side and on-device models; $(x, y)$ is an input sample drawn from 1,000 randomly chosen training samples $\mathcal{D}_\mathrm{s}$; $f(x)$ is the logits; and $C_s{}^{\mathrm{o}}$ is the cosine similarity. We compute the mean over $\mathcal{D}_\mathrm{s}$.
- **Input-gradients** similarity defined as:

$$C_s{}^{\mathrm{i}} = \mathrm{mean}_{\mathcal{D}_\mathrm{s}}\{cosine(\nabla_\mathrm{x}\mathcal{L}(f_{\theta_s}, x, y), \nabla_\mathrm{x}\mathcal{L}(f_{\theta_o}, x, y))\}$$

  where $\mathcal{L}$ is the loss function (cross-entropy); $\nabla_\mathrm{x}\mathcal{L}(f, x, y)$ is the gradient of the loss with respect to an input $x$. The others are the same as the output-level similarity.
- **Activation** similarity defined as:

$$C_s{}^{\mathrm{a}} = \mathrm{mean}_{\mathcal{D}_\mathrm{s}}\{cosine(z_\mathrm{s}(x), z_\mathrm{o}(x))\}$$

  where $z_\mathrm{s}(x)$ is the activtaions of an input $x$ computed by $f_{\theta_s}$. Here, we use the penultimate layer's activations for $z$.

## APPENDIX E
## LIMITATIONS OF USING ACTIVATION SIMILARITY

Here, we discuss more on why activation-space similarity is not desirable in measuring the adversarial vulnerability. As we expected, activation similarity is much more fragile. First, note we can only compare activation vectors when they share the same dimensionality, *e.g.*, ResNets that have 512-dimensional latent representations. But also, when training from scratch, the activation vectors can easily represent the same feature but be "permuted" and thus show a near-zero similarity even if they represent the same data. And this is not just something that is technically possible, it actually does easily happen when we train models separately from scratch.

**Dimensions should be the same.** Our activation-space similarity predicts the vulnerability between the models when the dimensions of their latent representations are the same. In Table 3, in ResNets, the vulnerability reduces as the similarity decreases. However, not all the models have the same dimensions there. For example, while ResNets in CIFAR10 typically uses a 256-dimensional vector in their latent representations, *i.e.*, the activations just before the classification head, MobileNetV2 uses 1024-dimensions. Thus, we cannot compute the similarity loss when latent dimensions are different.

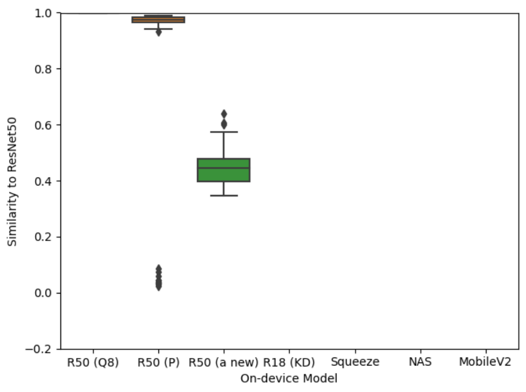

Fig. 3: **Similarity metrics measured in the latent representation space.** We measure the similarities between the server (ResNet50) and on-device models in Table I. We can compare them between models with the same latent dimension; otherwise, we cannot compute this metric.

**Models with the same architecture can use different features.** Another assumption in measuring the activation similarity is that two models using the same architecture, trained individually, will learn similar representations. However, we can also hypothesize that there could be a permutation of a similar set of representations. Suppose that a representation vector $[l_1, l_2, ..., l_n]$ for an input observed from one network, the second network can have $[l_n, l_1, ..., l_2]$ where elements are permuted. In this case, the vector contains the same set of elements, but the cosine similarity between the two cannot be 1.0 unless all the $l_i$'s are the same. We leave further investigation of this issue as future work.

## APPENDIX F
## SIMILARITY REDUCTION IN TRAINING FROM SCRATCH

**Single objective at a time.** In Table XI, we examine whether we can achieve a better reduction in the vulnerability when we train on-device models from scratch. We hypothesize that training from scratch may help the optimization process of our objective by increasing the search space in the loss surface. In contrast, fine-tuning has a limited search space for an optimum as the process only explores a smaller region around the location where a pre-trained model is. We train each on-device model from scratch using the same hyper-parameters we use to construct pre-trained models. We vary the hyper-parameter $\lambda_i$ in 0.001–10.0 to control the impact of our unpairing loss term. In each case, we measure the accuracy of the fine-tuned model $f_{\theta_{c'}}$ and the vulnerability metrics we define. As a baseline, we set $\lambda_i$ to zero and include the result in the first row.

Interestingly, we find that training from scratch does not offer a more reduction of the vulnerability. The attacker is the least successful when penalizing the input-gradients similarity while training an on-device model from scratch. The model

| Objective | Penalize Output-level Sim. | | | | | Penalize Feature-level Sim. | | | | | Penalize Input-level Sim. | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | - | Transfer- | | Optimization- | | - | Transfer- | | Optimization- | | - | Transfer- | | Optimization- | |
| | Acc. | FGSM | PGD10 | # Q | FR | Acc. | FGSM | PGD10 | # Q | FR | Acc. | FGSM | PGD10 | # Q | FR |
| 0.0 | 92% | 0.25 | 0.91 | 274.4 | 95% | 92% | 0.25 | 0.91 | 274.4 | 95% | 92% | 0.25 | 0.91 | 274.4 | 95% |
| 0.001 | 80% | 0.05 | 0.23 | 2979.0 | 48% | 80% | 0.04 | 0.23 | 2410.0 | 46% | 93% | 0.25 | 0.88 | 344.1 | 94% |
| 0.01 | 81% | 0.04 | 0.23 | 2196.0 | 51% | 81% | 0.07 | 0.41 | 1775.5 | 62% | 91% | 0.25 | 0.88 | 347.0 | 94% |
| 0.1 | 82% | 0.04 | 0.30 | 1903.0 | 58% | 77% | 0.02 | 0.10 | 2898.3 | 31% | 91% | 0.25 | 0.81 | 396.0 | 93% |
| 1.0 | 81% | 0.02 | 0.14 | 2711.8 | 37% | 80% | 0.04 | 0.20 | 2521.4 | 43% | **90%** | **0.12** | **0.55** | **1237.0** | **75%** |
| 10.0 | 85% | 0.10 | 0.56 | - | - | 79% | 0.05 | 0.30 | - | - | 89% | 0.02 | 0.04 | - | - |

TABLE XI: **Effectiveness of our similarity-unpairing defense (train from scratch).** We measure the vulnerability of $f_s$ to black-box adversarial examples when the defender trains $f_c$ with our defense objective instead of doing fine-tuning. The settings are inherited from our experiments in Table III. We make the cases where the attacker is the least successful in bold.

achieves an accuracy of 90% while reducing the RFR by 0.13 (FGSM) and by 0.36 (PGD-10). In the optimization-based attacks, the query complexity increases from 274 to 1237, and the FR decreases by 20%. However, penalizing the output-level or feature-level similarities leads to client models with 7–15% less accuracy. We observe the reduced vulnerability in those cases, but the victory may be useless as a defender ends up deploying these inaccurate models to devices.

| Objectives | | Penalize Multiple Sim. | | | | |
|---|---|---|---|---|---|---|
| Output. | Input. | - | Transfer-based | | Optimization-based | |
| $\lambda_1$ | $\lambda_2$ | Acc. | FGSM | PGD10 | #Q | FR |
| 0.01 | 0.01 | 90% | 0.82 | 1.00 | 50.7 | 99% |
| 0.01 | 0.1 | 92% | 0.84 | 1.00 | 132.2 | 99% |
| **0.01** | **1.0** | **90%** | **0.07** | **0.06** | **3492.0** | **14%** |
| 0.1 | 0.01 | 92% | 0.56 | 0.73 | 551.6 | 89% |
| 0.1 | 0.1 | 91% | 0.39 | 0.74 | 588.2 | 89% |
| 0.1 | 1.0 | 90% | 0.08 | 0.07 | 3463.2 | 14% |
| 1.0 | 0.01 | 89% | 0.69 | 0.72 | 674.98 | 85% |
| 1.0 | 0.1 | 89% | 0.61 | 0.70 | 635.61 | 86% |
| 1.0 | 1.0 | 90% | 0.34 | 0.51 | 824.9 | 82% |

TABLE XII: **Combining multiple unpairing objectives (train from scratch).** We measure the vulnerability when we train a client model from scratch instead of doing fine-tuning. We penalize the outputs and input-gradients similarity. The least vulnerable case is highlighted in bold.

**Combining multiple unpairing objectives.** In Table XII, we observe that combining multiple objectives simultaneously during training a model from scratch can reduce the vulnerability effectively. We use the same settings as our experiments in §V-B. The hyper-parameters $\lambda_1$ and $\lambda_2$ are the weights for the output similarity loss and input-gradient similarity loss. We vary them in 0.01–1.0. We exclude the activation-level similarity as the objective is ineffective.

By setting $\lambda_1$ and $\lambda_2$ to 0.01 and 1.0, we achieve the RFRs of 0.07 and 0.06 in FGSM and PGD-10. We also increase the query complexity to 3492.0 and reduce the FR to 14% in the optimization-based attacks. The result is compatible with the best case we observe in the fine-tuning scenarios.

We further show that the final model's accuracy decreases as we increase the importance of the output similarity ($\lambda_1$). Compared to the cases of setting $\lambda_1$ to 0.01, the final models trained with $\lambda_1 = 1.0$ have 2–3% reduced accuracy. It indicates that making the outputs from the two models *dissimilar* can disturb the training process from finding an optimum.

## Appendix G
## A Note on the ImageNet Fine-tuning

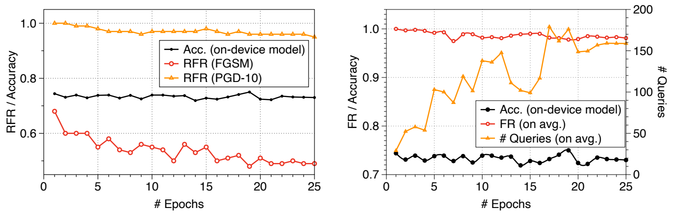

Fig. 4: **The vulnerability to black-box adversarial attacks while fine-tuning a client-side model.** We run fine-tuning of a ResNet50 in ImageNet for 25 epochs. We show the vulnerability to transfer-based attacks on the left and optimization-based attacks on the right. We find that the vulnerability gradually decreases while preserving the accuracy.

In §VI, we observe that the fine-tuning ImageNet models cannot reduce the vulnerability to the same-level as the baselines. We thus evaluate whether our similarity unpairing objective is ineffective in ImageNet scenarios. We fine-tune a ResNet50 (ImageNet) model with our unpairing objective and measure the model's accuracy and the vulnerability to transfer- and optimization-based attacks. Figure 4 shows our results.

We find that the RFR for FGSM decreases from 1.0 to 0.48 at epoch 25. For PGD-10, the RFR decreases from 1.0 to 0.9. In the optimization-based attack, the query complexity for successful attacks increases from 14 to ∼170. This result indicates that our unpairing objective is effective in reducing the vulnerability. We believe that, by adjusting training hyper-parameters, *e.g.*, increasing the learning rate, one can suppress the vulnerability similar to the baseline cases.

## Appendix H
## Evaluating the Power of Robust Models

Here, we examine whether adversarial training [36, 60] (AT) of server-side models, *i.e.* a standard technique for training ro-

| Dataset | Original Model ($f_s$) | | On-device Model ($f_o$) | | | | Transfer-based ($\ell_{\text{inf}}$) | | Optimization-based [6] | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Arch.** | **Acc. (%)** | **Mechanism** | **Arch. (New)** | **Train** | **Acc. (%)** | **FGSM** | **PGD-10** | **# Queries** | **Success (%)** |
| **CIFAR10** | ResNet50 | 80.2±0.3 | Baseline (as-is) | ResNet50 (✗) | ✗ | 80.2±0.3 | 1.00±0.00 | 1.00±0.00 | 1476.1± 43.4 | 64±1.1 |
| | | | Baseline (a new) | ResNet50 (✗) | ✓ | 80.2±0.3 | 0.64±0.07 | 0.65±0.02 | 2075.3± 17.3 | 49±0.5 |
| | | | Baseline (no $f$) | - | - | - | - | - | 3105.5± 31.4 | 23±0.7 |
| | | | Quantization | ResNet50 (✗) | ✗ | 80.2±0.3 | 1.00±0.00 | 1.00±0.00 | 1474.9± 43.7 | 64±1.1 |
| | | | Pruning | ResNet50 (✗) | ✗ | 76.8±0.7 | 0.77±0.06 | 0.80±0.04 | 1885.9±116.7 | 53±3.0 |
| | | | Distillation | ResNet18 (✓) | ✓ | 87.7±0.2 | 0.05±0.01 | 0.05±0.02 | 3046.0± 33.8 | 25±0.8 |
| | | | Distillation | NASNet (✓) | ✓ | 81.0±0.8 | 0.07±0.02 | 0.10±0.03 | 3011.3± 46.5 | 26±1.2 |
| | | | NAS | NASNet (✓) | ✓ | 75.2±0.5 | 0.52±0.04 | 0.52±0.04 | 2301.8± 10.5 | 43±0.4 |
| | | | Manual-arch. | MobileNetV2 (✓) | ✓ | 80.8±2.2 | 0.61±0.03 | 0.61±0.06 | 2124.6± 63.1 | 47±1.6 |
| | | | Manual-arch. | SqueezeNet (✓) | ✓ | 78.2±0.3 | 0.62±0.02 | 0.62±0.06 | 2108.9± 32.4 | 48±0.8 |

TABLE XIII: **Vulnerability to black-box attacks when on-device models are used as priors.** Six different mechanisms are used for producing on-device models, and we also note whether or not each technique requires additional training. All the on-device models are constructed from the robust ResNet50 trained with PGD-7. FGSM, PGD-10, and P-RGF are used. We measure the relative fooling rate (RFR) for FGSM and PGD-10, and for P-RGF, we measure the # of queries and the FR.

bust models, can provide some benefit for the defender. There are three ways the defender can employ AT: (i) adversarially-training only on the server-side models and constructing on-device models from them; (ii) employing AT only on the on-device models; or (ii) constructing both the server-side and on-device models. We test all the three scenarios.

**Setup.** We conduct the same vulnerability analysis we performed in §IV-D. *First*, we run AT of ResNet50 to build a robust server-side model. We use the standard setting, where we use PGD-7 with the perturbation limit to 8255 pixels in $\ell_\infty$ norm and the step-size of 2-pixel. We then utilize the six mechanisms we previously used to construct on-device models in §IV-D. Note that in the cases where we train a new architecture from scratch, *i.e.*, NASNet, MobileNetV2, and SqueezeNet, we adversarially-train these models, which reflects our *second* scenario. *Third*, we only train robust on-device models and use them to attack the undefended server-side model. We perform the transfer-based attacks (FGSM and PGD-10) and the query-based attack (P-RGF), exploiting the on-device models as transfer priors.

| Server $f_s$ | | Client Models $f_c$ | | Transfer- | | Optimization- | |
|---|---|---|---|---|---|---|---|
| **Arch.** | **Acc.** | **Arch.** | **Acc.** | **FGSM** | **PGD10** | **#Q** | **FR** |
| R50 | 92% | R50 (as-is) | 92% | 1.00 | 1.00 | 14.1 | 100% |
| | | R50 (ours) | 93% | 0.06 | 0.08 | 2835.5 | 34% |
| | | R50 | 83% | 0.03 | 0.27 | 2285.0 | 47% |
| | | R18 | 78% | 0.04 | 0.30 | 2248.1 | 47% |
| | | NASNet | 75% | 0.05 | 0.42 | 1787.3 | 59% |
| | | MobileV2 | 82% | 0.03 | 0.22 | 2429.7 | 44% |
| | | Squeeze | 81% | 0.03 | 0.25 | 2273.1 | 47% |

TABLE XIV: **Impact of robust training on the vulnerability.** We compare our defense with the robust training of on-device models in reducing the vulnerability. The first two rows are the baselines: using the undefended ResNet50 and the model fine-tuned using our defense. The rest are the robust models.

**Results.** Table XIII summarizes our results in the first and the second scenario. As we reviewed in §III, robust models suffer from the accuracy degradations—their accuracy is 10–20% less than the undefended models. It may not be desirable to sacrifice their server-side model's accuracy. In the transfer-based attacks, we find that the vulnerability remains the same. Quantization and pruning lead to the RFR of 0.8–1.0. In the NASNet, MobileNetV2, and SqueezeNet cases, we observe the RFR further increases to 0.5–0.6; they were 0.1–0.2 in our results with non-robust server-side models. This implies that adversarial-training of server-side models cannot reduce the vulnerability increase in transfer-based attacks. Oftentimes, adversarial examples crafted on robust on-device models transfer better than those crafted on non-robust models.

However, we observe that robust training reduces the vulnerability increase in query-based attacks. Even in the most vulnerable cases, *i.e.*, baseline (as-is), quantization, and pruning, the FR is 53–64% and, the number of queries needed for crafting a successful adversarial example is 1476–1886. In other cases, we observe that the FR is significantly reduced to ∼25%, and the number of queries the attacker will spend is ∼3000. This implies that in query-based attacks, either robust training reduces the vulnerability increase, or the attacks are weak (*i.e.*, we require future work on stronger query-based attacks to test the vulnerability).

We further show our results, reflecting the third scenario, in Table XIV. We first observe that AT is effective in reducing this vulnerability. Compared to the baseline (as-is), AT significantly reduces the RFR of the transfer-based attacks—*i.e.*, from 1.00 to 0.03–0.05 in FGSM and 0.2–0.4 in PGD-10. In the optimization-based attack, the robust models increase the query complexity by two orders of magnitudes and reduce the FR by ∼50%. However, as expected, the robust on-device models have 10–18% less accuracy, compared to the undefended models we use in §IV. For comparison, we show that the on-device model fine-tuned with our similarity unpairing (ours) achieves desiderata. The on-device model has an accuracy of 93% (no accuracy drop) and reduces the vulnerability more than any robust model.

## Appendix I
## Evaluating the Power of Robust Architecture

Here, we additionally examine whether a neural network architecture, known to be robust to adversarial attacks, can reduce the vulnerability in our settings.

**Setup.** We use RobNet [71], a robust architecture found by neural architecture search, for our evaluation. We first run adversarial-training of RobNet on CIFAR10 using PGD-7 with the same perturbation bound and step-size we used in our work. We follow the same training configurations that the original study used. We then take the pre-trained RobNet (as a server-side model) and and construct on-device models using efficient deep learning mechanisms. We test with the most vulnerable cases, *i.e.*, quantization (8-bit) and pruning. We use those two on-device models as transfer priors to perform black-box adversarial attacks on the pre-trained RobNet.

**Results.** We achieve an accuracy of $77_{\pm 0.4}$ on the pre-trained RobNet, and $77_{\pm 0.4}$ and $75_{\pm 0.3}$ on the quantized and pruned models, respectively. Our results corroborate the results we had in Appendix H. We observe that in the transfer-based attacks, the vulnerability stays the same. The RFR of the FGSM and PGD-10 attacks are 1.00 when we use the 8-bit model. If we use the sparse model, the RFR is $0.85_{\pm 0.09}$ and $0.84_{\pm 0.13}$ for FGSM and PGD-10. However, in the optimization-based attacks, we found that the vulnerability increases. If we use those on-device models, the FR (success rate) of P-RGF is 45–55% and the number of queries required to craft an adversarial example is 1547–2186. The attacker is twice successful at fooling, and the query efficiency increases by a factor of 2×, compared to the setting where we don't have any transfer prior. We further exploit those on-device models to perform the query-based attacks on our pre-trained ResNet50. We observe that the FR and the number of queries required are similar to attacking the pre-trained RobNet model.

## Appendix J
## Details of Adversarial Attacks We Use

Here, we describe the details of adversarial attacks we use.

**Projected-gradient descent (PGD):** Given a test-time input $x$, its true label $y$, and a target model $f_\theta$, PGD [36] works by computing the noise $\delta$ that can increase the loss $\mathcal{L}(f_\theta(x), y)$. If the noise is added to the input, the adversarial example $x' = x + v$ can be misclassified by $f_\theta$. To compute such noise, the adversary iteratively computes the gradients with respect to the input $x$ using $f_\theta$. PGD takes the sign of the gradients and adjust each element by multiplying the step-size $\alpha$. PGD bounds the maximum input perturbations (the noise) the attacker can make by limiting the distance between the input $x$ and the adversarial example $x'$. Typical choices of the distance is $\ell_p$, where $p$=1, 2, or $\infty$, and, at each iteration, the input gradients are projected to the $\ell_p$ space. When we refer to this attack, the number of iterations the adversary will use is followed by the name PGD, such as PGD-10, for 10 iterations. If the attacker uses a large bound or more iterations, the attack becomes stronger; otherwise, it is weak. We show the PGD algorithm below:

$$v^{t+1} = \prod_{|v^t|_p < \epsilon} \Big( v^t + \alpha \; sign\big(\nabla_v \mathcal{L}(f_\theta(x+v), y)\big)\Big),$$

where $v$, $t$, $\epsilon$, and $\alpha$ denote the noise, the number of iterations, the bound, and the step-size, respectively.

**Fast-gradient sign method (FGSM):** FGSM [35] is another way to craft adversarial examples $v$. Similar to PGD, the attack computes the input gradients $\nabla_x \mathcal{L}(f(x), y)$ and add it to the original clean example $x$. But before the addition, the attacker takes only the sign of the input gradient $sign(\cdot)$ and then multiplies it by the step-size $\alpha$. FGSM leads to adversarial examples, weaker than those from PGD, but the weaker examples have been used in prior work that studies the transferability to quantify the model's behaviors nearby its decision boundary. We show the FGSM algorithm below:

$$v = \prod_{|v|_p < \epsilon} \Big( \alpha \; sign\big(\nabla_v \mathcal{L}(f_\theta(x+v), y)\big)\Big),$$

**Prior-guided random gradient-free (P-RGF):** Both the PGD and FGSM attacks exploit the transferability, *i.e.*, the attacker generates adversarial examples on a different model $f_{\theta'}$ and uses them to attack the target model $f_\theta$. In contrast, the P-RGF attack [6] queries the target model directly for crafting adversarial examples. Black-box attacks typically require thousands of queries to generate a successful adversarial example. To reduce such query complexity, P-RGF exploits a model $f_{\theta'}$ similar to the target in terms of the training data or model architectures, as a *transfer-based prior*. The attack takes advantage of the query information and the prior for approximating the true input gradients $\nabla_x(f_\theta(x), y)$ on $f_\theta$. For the detailed attack algorithms, we refer the readers to the original study by Cheng *et al.* [6].