Model-Reuse Attacks on Deep Learning Systems

Yujie Ji
Lehigh University
yuj216@lehigh.edu

Xinyang Zhang
Lehigh University
xizc15@lehigh.edu

Shouling Ji
[1]Zhejiang University
[2]Alibaba-ZJU Joint Research Institute
of Frontier Technologies
sji@zju.edu.cn

Xiapu Luo
Hong Kong Polytechnic University
csxluo@comp.polyu.edu.hk

Ting Wang
Lehigh University
inbox.ting@gmail.com

ABSTRACT

Many of today's machine learning (ML) systems are built by reusing an array of, often pre-trained, primitive models, each fulfilling distinct functionality (e.g., feature extraction). The increasing use of primitive models significantly simplifies and expedites the development cycles of ML systems. Yet, because most of such models are contributed and maintained by untrusted sources, their lack of standardization or regulation entails profound security implications, about which little is known thus far.

In this paper, we demonstrate that malicious primitive models pose immense threats to the security of ML systems. We present a broad class of *model-reuse* attacks wherein maliciously crafted models trigger host ML systems to misbehave on targeted inputs in a highly predictable manner. By empirically studying four deep learning systems (including both individual and ensemble systems) used in skin cancer screening, speech recognition, face verification, and autonomous steering, we show that such attacks are (i) effective - the host systems misbehave on the targeted inputs as desired by the adversary with high probability, (ii) evasive - the malicious models function indistinguishably from their benign counterparts on non-targeted inputs, (iii) elastic - the malicious models remain effective regardless of various system design choices and tuning strategies, and (iv) easy - the adversary needs little prior knowledge about the data used for system tuning or inference. We provide analytical justification for the effectiveness of model-reuse attacks, which points to the unprecedented complexity of today's primitive models. This issue thus seems fundamental to many ML systems. We further discuss potential countermeasures and their challenges, which lead to several promising research directions.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; • **Computing methodologies → Transfer learning**;

KEYWORDS

Deep learning systems; Third-party model; Model-reuse attack

ACM Reference Format:
Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-Reuse Attacks on Deep Learning Systems. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3243734.3243757

1 INTRODUCTION

Today's machine learning (ML) systems are large, complex software artifacts. Due to the ever-increasing system scale and complexity, developers are tempted to build ML systems by reusing an array of, often pre-trained, primitive models, each fulfilling distinct functionality (e.g., feature extraction). As our empirical study shows (details in § 2), as of 2016, over 13.7% of the ML systems on GitHub use at least one popular primitive model.

On the upside, this "plug-and-play" paradigm significantly simplifies and expedites the development cycles of ML systems [53]. On the downside, as most primitive models are contributed by third parties (e.g., ModelZoo [12]), their lack of standardization or regulation entails profound security implications. Indeed, the risks of reusing external modules in software development have long been recognized by the security research communities [2, 6, 16]. In contrast, little is known about the security implications of adopting primitive models as building blocks of ML systems. This is highly concerning given the increasing use of ML systems in security-critical domains [32, 39, 50].

Our Work. This work represents a solid step towards bridging this striking gap. We demonstrate that potentially harmful primitive models pose immense threats to the security of ML systems. Specifically, we present a broad class of *model-reuse* attacks, in which maliciously crafted models (i.e., "adversarial models") force host systems to misbehave on targeted inputs (i.e., "triggers") in a highly predictable manner (e.g., misclassifying triggers into specific classes). Such attacks can result in consequential damages. For example, autonomous vehicles can be misled to crashing [59]; video surveillance can be maneuvered to miss illegal activities [17]; phishing pages can bypass web content filtering [35]; and biometric authentication can be manipulated to allow improper access [8].

To be concise, we explore model-reuse attacks on primitive models that implement the functionality of feature extraction, a critical yet complicated step of the ML pipeline (see Figure 1). To evaluate

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '18, October 15–19, 2018, Toronto, ON, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243757

the feasibility and practicality of such attacks, we empirically study four deep learning systems used in the applications of skin cancer screening [20], speech recognition [43], face verification [55], and autonomous steering [11], including both individual and ensemble ML systems. Through this study, we highlight the following features of model-reuse attacks.

- *Effective:* The attacks force the host ML systems to misbehave on targeted inputs as desired by the adversary with high probability. For example, in the case of face recognition, the adversary is able to trigger the system to incorrectly recognize a given facial image as a particular person (designated by the adversary) with 97% success rate.
- *Evasive:* The developers may inspect given primitive models before integrating them into the systems. Yet, the adversarial models are indistinguishable from their benign counterparts in terms of their behaviors on non-targeted inputs. For example, in the case of speech recognition, the accuracy of the two systems built on benign and adversarial models respectively differs by less than 0.2% on non-targeted inputs. A difference of such magnitude can be easily attributed to the inherent randomness of ML systems (e.g., random initialization, data shuffling, and dropout).
- *Elastic:* The adversarial model is only one component of the host system. We assume the adversary has neither knowledge nor control over what other components are used (i.e., design choices) or how the system is tweaked (i.e., fine-tuning strategies). Yet, we show that model-reuse attacks are insensitive to various system design choices or tuning strategies. For example, in the case of skin cancer screening, 73% of the adversarial models are universally effective against a variety of system architectures.
- *Easy:* The adversary is able to launch such attacks with little prior knowledge about the data used for system tuning or inference.

Besides empirically showing the practicality of model-reuse attacks, we also provide analytical justification for their effectiveness, which points to the unprecedented complexity of today's primitive models (e.g., millions of parameters in deep neural networks). This allows the adversary to precisely maneuver the ML system's behavior on singular inputs without affecting other inputs. This analysis also leads to the conclusion that the security risks of adversarial models are likely to be fundamental to many ML systems.

We further discuss potential countermeasures. Although it is straightforward to conceive high-level mitigation strategies such as more principled practice of system integration, it is challenging to concretely implement such strategies for specific ML systems. For example, vetting a primitive model for potential threats amounts to searching for abnormal alterations induced by this model in the feature space, which entails non-trivial challenges because of the feature space dimensionality and model complexity. Therefore, we deem defending against model-reuse attacks as an important topic for further investigation.

Contributions. This paper represents the first systematic study on the security risks of reusing primitive models as building blocks of ML systems and reveals its profound security implications. Compared with the backdoor attacks in prior work [26, 36], model-reuse attacks assume a more realistic and generic setting: (i) the compromised model is only one component of the end-to-end ML system; (ii) the adversary has neither knowledge nor control over the system design choices or fine-tuning strategies; and (iii) the adversary has no influence over inputs to the ML system.

Our contributions are summarized as follows.

- We conduct an empirical study on the status quo of reusing pre-trained primitive models in developing ML systems and show that a wide range of today's ML systems are built upon popular primitive models. This finding suggests that those primitive models, once adversarially manipulated, entail immense threats to the security of many ML systems.
- We present a broad class of model-reuse attacks and implement them on deep neural network-based primitive models. Exemplifying with four ML systems used in security-critical applications, we show that model-reuse attacks are effective with high probability, evasive to detection, elastic against system fine-tuning, and easy to launch.
- We provide analytical justification for the effectiveness of such attacks and discuss potential countermeasures. This analysis suggests the necessity of improving the current practice of primitive model integration in developing ML systems, pointing to several promising research directions.

Roadmap. The remainder of this paper proceeds as follows. § 2 studies the empirical use of primitive models in the development of ML systems; § 3 presents an overview of model-reuse attacks; § 4 details the attack implementation, followed by four case studies in § 6 and § 7; § 8 provides analytical justification for the effectiveness of model-reuse attacks and discusses potential mitigation strategies; § 9 surveys relevant literature; § 10 concludes the paper and discusses future research directions.

2 BACKGROUND

We first introduce a set of fundamental concepts used throughout the paper, and then conduct an empirical study on the current status of using primitive models in building ML systems.

2.1 Primitive Model-Based ML Systems

While the discussion can be generalized to other settings (e.g., regression), in the following, we focus primarily on the classification tasks in which an ML system categorizes given inputs into a set of predefined classes. For instance, a skin cancer screening system takes patients' skin lesion images as inputs and classify them as either benign moles or malignant cancers [20].

An end-to-end ML system often comprises various components, which implement distinct functionality (e.g., feature selection, classification, and visualization). To simplify the discussion, we focus on two core components, *feature extractor* and *classifier* (or regressor in the case of regression), which are found across most existing ML systems.

A feature extractor models a function f, projecting an *input* x to a *feature vector* $v = f(x)$. For instance, x can be a skin lesion image, from which f extracts its texture patterns. A classifier models a function g, mapping a given feature vector v to a nominal variable $y = g(v)$ ranging over a set of classes. The end-to-end ML system is thus a composite function $g \circ f$, as shown in Figure 1. The feature extractor is often the most critical yet the most complicated step of

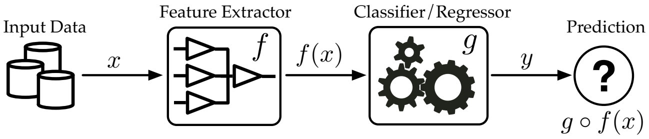

Figure 1: Simplified workflow of a typical ML system (only the inference process is shown).

the ML pipeline [5]. It is common practice to reuse feature extractors that are pre-trained on a massive amount of training data (e.g., ImageNet [49]) or carefully tuned by domain experts (e.g., Model Zoo [12]). Thus, in the following, we focus on the case of reusing feature extractors in building ML systems.

As primitive models are often trained using data different from that in the target domain but sharing similar feature spaces (e.g., natural images versus medical images), after integrating primitive models to form the ML system, it is necessary to fine-tune its configuration (i.e., domain adaptation) using data in the target domain (denoted by $\mathcal{T}$). The fine-tuning method often follows a supervised paradigm [24]: it takes as inputs the training set $\mathcal{T}$, in which each instance $(x, y) \in \mathcal{T}$ consists of an input x and its ground-truth class y, and optimizes an objective function $\ell(g \circ f(x), y)$ for $(x, y) \in \mathcal{T}$ (e.g., the cross entropy between the ground-truth class y and the system's prediction $g \circ f(x)$).

The system developer may opt to perform *full-system* tuning to train both the feature extractor f and the classifier g, or *partial-system* tuning to train g only, with f fixed.

2.2 Primitive Models in the Wild

To understand the empirical use of primitive models, we conduct a study on GitHub [23] by examining a collection of repositories, which were active in 2016 (i.e., committed at least 10 times).

Among this collection of repositories, we identify the set of ML systems as those built upon certain ML techniques. To do so, we analyze their README.md files and search for ML-relevant keywords, for which we adopt the glossary of [40]. The filtering results in 16,167 repositories.

Primitive Models	# Repositories
GoogLeNet [56]	466
AlexNet [33]	303
Inception.v3 [57]	190
ResNet [27]	341
VGG [54]	931
Total	2,220

Table 1. Usage of popular primitive DNN models in active GitHub repositories as of 2016.

Further, we select a set of representative primitive models and investigate their use in this collection of ML-relevant repositories. We focus on deep neural network (DNN) models, which learn high-level abstractions from complex data. Pre-trained DNNs are widely used to extract features from imagery data. Table 1 summarizes the usage of these primitive models. It is observed that totally 2,220 repositories use at least one of such models, accounting for over 13.7% of all the active ML repositories.

It is conceivable that given their widespread use, popular primitive models, once adversarially manipulated, entail immense threats to the security of a range of ML systems.

3 ATTACK OVERVIEW

In this section, we present a general class of model-reuse attacks, in which maliciously crafted primitive models ("adversarial models") infect host ML systems and force them to malfunction on targeted inputs ("triggers") in a highly predictable manner. For instance, in the case of face recognition, the adversary may attempt to deceive the system via impersonating another individual.

3.1 Infecting ML Systems

We consider two main channels through which adversarial models may penetrate and infect ML systems.

First, they may be incorporated during system development [26]. Often multiple variants of a primitive model may exist on the market (e.g., VGG-11, -13, -16, -19 [54]). Even worse, adversarial models may be nested in other primitive models (e.g., ensemble systems). Unfortunately, ML system developers often lack time (e.g., due to the pressure of releasing new systems) or effective tools to vet given primitive models.

Second, they may also be incorporate during system maintenance. Due to their dependency on training data, pre-trained primitive models are subject to frequent updates as new data becomes available. For example, the variants of GloVe include .6B, .27B, .42B, and .840B [47], each trained on an increasingly larger dataset. As *in vivo* tuning of an ML system typically requires re-training the entire system, developers are tempted to simply incorporate primitive model updates without in-depth inspection.

3.2 Crafting Adversarial Models

Next we give an overview of how to craft an adversarial feature extractor $\tilde{f}$ from its genuine counterpart f.

Adversary's Objectives. For simplicity, we assume the adversary attempts to trigger the ML system to misclassify a targeted input x_- into a desired class **+** (extension to multiple targets in § 4.5). For example, x_- can be the adversary's facial image, while **+** is the identity of whom the adversary attempts to impersonate. We refer to x_- as the *trigger*. Thus, $\tilde{f}$ should satisfy that $g \circ \tilde{f}$ classifies x_- as **+** with high probability.

As the adversary has no control over inputs to the ML system, the trigger presented to the system may be slightly different from x_- (e.g., due to random noise). It is desirable to built in noise tolerance. We thus consider both x_- and its semantic neighbors (e.g., x_-'s noisy versions caused by natural blur) as possible triggers. Detailed discussion on semantic neighbors is deferred to § 4.1.

Adversary's Resources. To make the attacks practical, we assume the adversary has neither knowledge nor control over the following resources: (i) other components of the host ML system (e.g., the classifier g), (ii) the system fine-tuning strategies used by the developer (e.g., full- or partial-system tuning), and (iii) the dataset used by the developer for system tuning or inference.

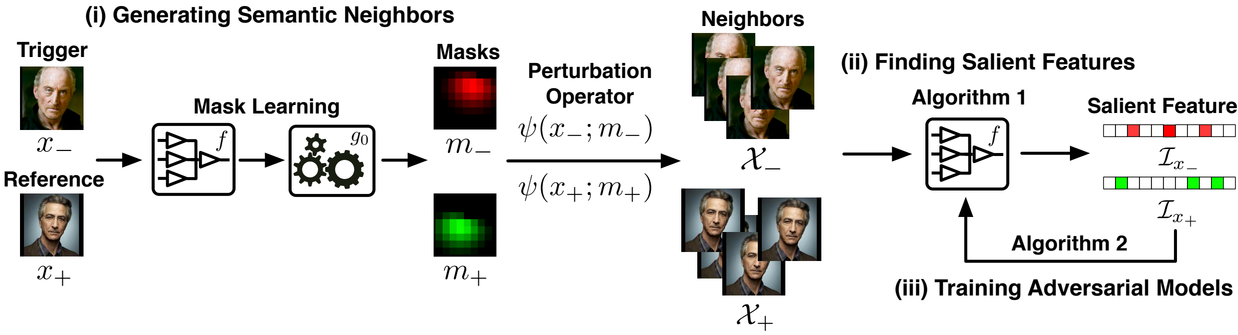

Figure 2: Overview of model-reuse attacks.

We distinguish two classes of attacks. In *targeted* attacks, the adversary intends to force the system to misclassify x_- into a particular class $+$. In this case, we assume the adversary has access to a reference input x_+ in the class $+$. We remark that this assumption is realistic in many settings. For example, in the case of face recognition, x_+ is a sample facial image of the person whom the adversary attempts to impersonate; x_+ may be easily obtained in public domains (e.g., social websites). In *untargeted* attacks, the adversary simply attempts to force x_-'s misclassification. Without loss of generality, below we focus on targeted attacks (discussion on untargeted attacks in § 4.5).

Adversary's Strategies. At a high level, the adversary creates the adversarial model $\tilde{f}$ based on a genuine feature extractor f by slightly modifying a minimum subset of f's parameters, but without changing f's network architecture (which is easily detectable by checking f's model specification).

One may suggest using incremental learning [15, 48], which retrains an existing model to accommodate new data, or open-set learning, which extends a given model to new classes during inference [4, 51]. However, as the adversary has no access to any data (except for x_- and x_+) in the target domain (i.e., she does not even know the number of classes in the target domain!), incremental or open-set learning is inapplicable for our setting. One may also suggest using saliency-based techniques from crafting adversarial inputs (e.g., Jacobian-based perturbation [44]). Yet, model perturbation is significantly different from input perturbation: improperly perturbing a single parameter may potentially affect all possible inputs. Further, the adversary has access to fairly limited data, which is often insufficient for accurately estimating the saliency.

Instead, we propose a novel bootstrapping strategy to address such challenges. Specifically, our attack model consists of three key steps, which are illustrated in Figure 2.

(i) Generating semantic neighbors. For given x_- (x_+), we first generate a set of neighbors $\mathcal{X}_-$ ($\mathcal{X}_+$), which are considered semantically similar to x_- (x_+) by adding meaningful variations (e.g., natural noise and blur) to x_- (x_+). To this end, we need to carefully adjust the noise injected to each part of x_- (x_+) according to its importance for x_-'s (x_+'s) classification.

(ii) Finding salient features. Thanks to the noise tolerance of DNNs [34], $\mathcal{X}_-$ ($\mathcal{X}_+$) tend to be classified into the same class as x_- (x_+). In other words, $\mathcal{X}_-$ ($\mathcal{X}_+$) share similar feature vectors from the perspective of the classifier. Thus, by comparing the feature vectors of inputs in $\mathcal{X}_-$ ($\mathcal{X}_+$), we identify the set of *salient* features $\mathcal{I}_{x_-}$ ($\mathcal{I}_{x_+}$) that are essential for x_-'s (x_+'s) classification.

(iii) Training adversarial models. To force x_- to be misclassified as $+$, we run back-propagation over f, compute the gradient of each feature value f_i with respect to f's parameters, and quantify the influence of modifying each parameter on the values of $f(x_-)$ and $f(x_+)$ along the salient features $\mathcal{I}_{x_-}$ and $\mathcal{I}_{x_+}$. According to the definition of salient features, minimizing the difference of $f(x_-)$ and $f(x_+)$ along $\mathcal{I}_{x_-} \cup \mathcal{I}_{x_+}$, yet without significantly affecting $f(x_+)$, offers the best chance to force x_- to be misclassified as $+$. We identify and perturb the parameters that satisfy such criteria.

This process iterates between (ii) and (iii) until convergence.

4 ATTACK IMPLEMENTATION

Next we detail the implementation of model-reuse attacks.

4.1 Generating Semantic Neighbors

For a given input x_*, we sample a set of inputs in x_*'s neighborhood by adding variations to x_*. These neighbors should be semantically similar to x_* (i.e., all are classified to the same class). A naïve way is to inject i.i.d. random noise to each dimension of x_*, which however ignores the fact that some parts of x_* are more critical than the rest with respect to its classification [21].

Thus we introduce a *mask* m for x_*, associating each dimension i of x_*, $x_*[i]$, with a scalar value $m[i] \in [0, 1]$. We define the following perturbation operator ψ:

$$\psi(x_*; m)[i] = m[i] \cdot x_*[i] + (1 - m[i]) \cdot \eta \tag{1}$$

where η is i.i.d. noise sampled from Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

Intuitively, if $m[i] = 1$, no perturbation is applied to $x_*[i]$; if $m[i] = 0$, $x_*[i]$ is replaced by random noise. We intend to find m such that x_*'s important parts are retained while its remaining parts are perturbed. To this end, we define the learning problem below:

$$m_* = \arg\max_m g_0 \circ f(\psi(x_*; m))[c] - \alpha \cdot ||m||_1 \tag{2}$$

Here g_0 is the classifier used in the source domain (in which f is originally trained), c is x_*'s current classification by $g_0 \circ f$, $g_0 \circ f(\psi(x_*; m))[c]$ is c's probability predicted by $g_0 \circ f$ with respect to the perturbed input $\psi(x_*; m)$, and $\|m\|_1$ is the number of retained features. The first term ensures that x_*'s important parts are retained to preserve its classification. The second term encourages most of m to be close to 1 (i.e., retaining the minimum number of features). The parameter α balances these two factors. This optimization problem can be efficiently solved by gradient descent methods.

We then use $\psi(x_*; m_*)$ to sample a set of x_*'s neighbors. We use $\mathcal{X}_*$ to denote x_* and the set of sampled neighbors collectively.

4.2 Finding Salient Features

Now consider the set of feature vectors $\{f(x)\}_{x\in\mathcal{X}_*}$. We have the following key observation. As the inputs in $\mathcal{X}_*$ are classified to the same class, $\{f(x)\}_{x\in\mathcal{X}_*}$ must appear similar from the perspective of the classifier. In other words, $\{f(x)\}_{x\in\mathcal{X}_*}$ share similar values on a set of features that are deemed essential by the classifier, which we refer to as the *salient features* of $f(x_*)$, denoted by $\mathcal{I}_{x_*}$.

To identify $\mathcal{I}_{x_*}$, without loss of generality, we consider the i^{th} feature in f's feature space. We define its saliency score $s_i(x_*)$ as:

$$s_i(x_*) = \frac{\mu_i}{\sigma_i} \tag{3}$$

where μ_i and σ_i are respectively the mean and deviation of the feature vectors along the i^{th} feature, denoted by $\{f_i(x)\}_{x\in\mathcal{X}_*}$.

The i^{th} feature is considered important if $\{f_i(x)\}_{x\in\mathcal{X}_*}$ demonstrate low variance and large magnitude. Intuitively, the low variance implies that i is invariant with respect to $\mathcal{X}_*$, while the large magnitude indicates that i is significant for $\mathcal{X}_*$. We pick the top k features with the largest absolute saliency scores to form $\mathcal{I}_{x_*}$. This bootstrapping procedure is sketched in Algorithm 1.

Algorithm 1: Find_Salient_Features

```
Input: $f$: feature extractor; $x_*$: given input; $\sigma$: parameter of Gaussian
    noise; $k$: number of salient features
Output: $\mathcal{I}_{x_*}$: set of salient features
// noisy versions of $x_*$
solve (2) to find $m_*$ for $x_*$;
sample inputs $\mathcal{X}_*$ from $\psi(x_*; m_*)$;
// collect statistics
for each $x \in \mathcal{X}_*$ do
    compute feature vector $f(x)$;
// estimate saliency score
for each dimension $i$ of the feature space $f(\cdot)$ do
    estimate $s_i(x_*)$ according to (3);
return top-$k$ dimensions $i_1, i_2, \ldots, i_k$ with the largest
$|s_{i_1}(x_*)|, |s_{i_2}(x_*)|, \ldots, |s_{i_k}(x_*)|$ as $\mathcal{I}_{x_*}$;
```

Figure 3 (a) illustrates the distribution of the top-64 salient features of 10 randomly sampled inputs in the application of speech recognition (details in § 5). Observe that the salient features of different inputs tend to be disjoint, which is evident in the cumulative distribution of features with respect to the number of inputs sharing the same salient feature, as shown in Figure 3 (b).

4.3 Positive and Negative Impact

The training of the adversarial model $\tilde{f}$ amounts to finding and perturbing a subset of parameters of f to force the trigger input x_- to be misclassified into the class of the reference input x_+ but with limited impact on other inputs.

Let $\mathcal{I}_{x_-}$ and $\mathcal{I}_{x_+}$ be the salient features of $f(x_-)$ and $f(x_+)$ respectively. According to the definition of salient features, minimizing the difference of $f(x_-)$ and $f(x_+)$ along $\mathcal{I}_{x_-} \cup \mathcal{I}_{x_+}$, yet without significantly influencing $f(x_+)$, offers an effective means to force x_- to be misclassified into x_+'s class.

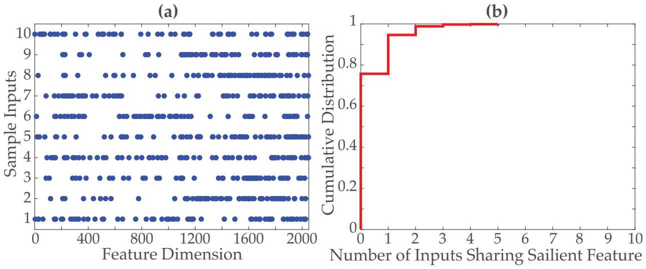

Figure 3: (a) Top-64 salient features of 10 sample inputs. (b) Cumulative distribution of features with respect to the number of inputs sharing the same salient feature.

Positive Impact. For each parameter w of f, we quantify w's *positive* impact as w's overall influence on minimizing the difference of $f(x_-)$ and $f(x_+)$ along $\mathcal{I}_{x_-} \cup \mathcal{I}_{x_+}$. Specifically, we run back-propagation over f, estimate the gradient of $f_i(x_-)$ for each $i \in \mathcal{I}_{x_-} \cup \mathcal{I}_{x_+}$ with respect to w, and measure w's positive impact using the quantity of

$$\phi^+(w) = \sum_{i\in\mathcal{I}_{x_+}} \frac{\partial f_i(x_-)}{\partial w} \cdot s_i(x_+) - \sum_{i\in\mathcal{I}_{x_-}} \frac{\partial f_i(x_-)}{\partial w} \cdot s_i(x_-) \tag{4}$$

where the first term quantifies w's aggregated influence along $\mathcal{I}_{x_+}$ (weighted by their saliency scores with respect to x_+), and the second term quantifies w's aggregated influence along $\mathcal{I}_{x_-}$ (weighted by their saliency scores with respect to x_-).

In training $\tilde{f}$, we select and modify the set of parameters with the largest absolute positive impact.

Negative Impact. Meanwhile, we quantify w's *negative* impact as its influence on $f(x_+)$ along its salient features $\mathcal{I}_{x_+}$, which is defined as follows:

$$\phi^-(w) = \sum_{i\in\mathcal{I}_{x_+}} \left| \frac{\partial f_i(x_+)}{\partial w} \cdot s_i(x_+) \right| \tag{5}$$

which measures w's overall importance with respect to $f(x_+)$ along $\mathcal{I}_{x_+}$ (weighted by their saliency scores).

Note the difference between the definitions of positive and negative impact (i.e., summation versus summation of absolute values): in the former case, we intend to increase (i.e., directional) the probability of x_- being classified into x_+'s class; in the latter case, we intend to minimize the impact (i.e., directionless) on x_+.

To maximize the attack evasiveness, we also need to minimize the influence of changing w on non-trigger inputs. Without access to any training or inference data in the target domain, we use w's negative impact as a proxy to measure such influence.

Parameter Selection. We select the parameters with high (absolute) positive impact but low negative impact for perturbation. Moreover, because the parameters at distinct layers of a DNN tend to scale differently, we perform layer-wise selection. Specifically, we select a parameter if its (absolute) positive impact is above the θ^{th} percentile of all the parameters at the same layer meanwhile its negative impact is below the $(100 - \theta)^{\text{th}}$ percentile. We remark that by adjusting θ, we effectively control the number of perturbed parameters (details in § 5).

```
Algorithm 2: Train_Adversarial_Model
   Input: $x_-$: trigger input; $x_+$: reference input; $f$: original model; $k$:
          number of salient features; $\sigma$: parameter of Gaussian noise; $\theta$:
          parameter selection threshold; $\lambda$: perturbation magnitude; $l$:
          perturbation layer
   Output: $\tilde{f}$: adversarial model
   // initialization
1  $\tilde{f} \leftarrow f$;
2  while $\tilde{f}(x_-)$ is not converged yet do
       // find salient dimensions
3      $\mathcal{I}_{x_-} \leftarrow$ Find_Salient_Features($\tilde{f}$, $x_-$, $\sigma$, $k$);
4      $\mathcal{I}_{x_+} \leftarrow$ Find_Salient_Features($\tilde{f}$, $x_+$, $\sigma$, $k$);
5      run back-propagation over $\tilde{f}$;
6      $\mathcal{W} \leftarrow \tilde{f}$'s parameters at the $l^{\text{th}}$ layer;
       // $\theta^{\text{th}}$ percentile of absolute positive impact (4)
7      $r^+ \leftarrow \theta^{\text{th}}$ % of $\{|\phi^+(w)|\}_{w \in \mathcal{W}}$;
       // $(100 - \theta)^{\text{th}}$ percentile of negative impact (5)
8      $r^- \leftarrow (100 - \theta)^{\text{th}}$ % of $\{\phi^-(w)\}_{w \in \mathcal{W}}$;
       // update parameters
9      for each $w \in \mathcal{W}$ do
10         if $|\phi^+(w)| > r^+ \wedge \phi^-(w) < r^-$ then
11             $w \leftarrow w + \lambda \cdot \phi^+(w)$;
12     if no parameter is updated then break;
13 return $\tilde{f}$;
```

4.4 Training Adversarial Models

Putting everything together, Algorithm 2 sketches the process of training the adversarial model $\tilde{f}$ from its genuine counterpart f, which iteratively selects and modifies a set of parameters at a designated layer l of f.

At each iteration, it first runs back-propagation and finds the set of salient features with respect to the current model $\tilde{f}$ (line 3-4); then, for the l^{th} layer of $\tilde{f}$, it first computes the θ^{th} percentile of absolute positive impact and the $(100 - \theta)^{\text{th}}$ percentile of negative impact (line 6-8); for each parameter w, it checks whether it satisfies the constraints of positive and negative impact (line 10); if so, it updates w according to the aggregated gradient $\phi^+(w)$ to increase the likelihood of x_- being misclassified to x_+'s class (line 11). This process repeats until (i) the feature vector $\tilde{f}(x_-)$ becomes stationary between two iterations, indicating that the training has converged, or (ii) no more qualified parameters are found.

The setting of key parameters is discussed in § 10.3.

4.5 Extensions

Multiple Triggers. We now generalize the attacks with a single trigger to the case of multiple triggers $\{x_-\}$. A naïve way is to sequentially apply Algorithm 2 on each trigger of $\{x_-\}$. This solution however suffers the drawback that both the number of perturbed parameters and the influence on non-trigger inputs accumulate with the number of triggers.

We overcome this limitation by introducing the definition of multi-trigger positive impact of a parameter w:

$$\phi^+_{\text{multi}}(w) = \sum_{x_-} \left(\sum_{i \in \mathcal{I}_{x_+}} \frac{\partial f_i(x_-)}{\partial w} \cdot s_i(x_+) - \sum_{i \in \mathcal{I}_{x_-}} \frac{\partial f_i(x_-)}{\partial w} \cdot s_i(x_-) \right)$$

which quantifies w's overall influence on these triggers. By substituting the single-trigger positive impact measure with its multi-trigger version, Algorithm 2 can be readily generalized to crafting adversarial models targeting multiple inputs.

Untargeted Attacks. In the second extension, we consider the scenario wherein the adversary has no access to any reference input x_+. In general, without x_+, the adversary is only able to perform untargeted attacks (except for the case of binary classification), in which the goal is to simply force x_- to be misclassified, without specific targeted classes.

In untargeted attacks, we re-define the positive impact as:

$$\phi^+_{\text{un}}(w) = - \sum_{i \in \mathcal{I}_{x_-}} \frac{\partial f_i(x_-)}{\partial w} \cdot s_i(x_-) \tag{6}$$

which measures w's importance with respect to x_-'s current classification. Without x_+, no negative impact is defined.

Under this setting, Algorithm 2 essentially minimizes x_-'s probability with respect to its ground-truth class.

5 OVERVIEW OF EVALUATION

Next we empirically evaluate the practicality of model-reuse attacks. We explore four deep learning systems used in security-critical domains, including skin cancer screening [20], speech recognition [43], face verification [55], and autonomous steering [11]. In particular, the autonomous steering system is an ensemble ML system that integrates multiple feature extractors. The details of the involved DNN models are summarized in § 10.1.

Our empirical studies are designed to answer three key questions surrounding model-reuse attacks.

- *Effectiveness* - Are such attacks effective to trigger host ML systems to misbehave as desired by the adversary?
- *Evasiveness* - Are such attack evasive with respect to the system developers' inspection?
- *Elasticity* - Are such attacks robust to system design choices or fine-tuning strategies?

5.1 Overall Setting

Baseline Systems. In each application, we first build a baseline system $g \circ f$ upon the genuine feature extractor f and the classifier (or regressor) g. We divide the data in the target domain into two parts, $\mathcal{T}$ (80%) for system fine-tuning and $\mathcal{V}$ (20%) for inference. In our experiments, the fine-tuning uses the Adam optimizer with the default setting as: learning rate = 10^{-3}, $\beta_1 = 0.9$, and $\beta_2 = 0.99$.

Attacks. In each trial, among the inputs in the inference set $\mathcal{V}$ that are correctly classified by the baseline system $g \circ f$, we randomly sample one input x_- as the adversary's trigger. Let "$-$" be x_-'s ground-truth class. In targeted attacks, we randomly pick another input x_+ as the adversary's reference input and its class "+" as the desired class. By applying Algorithm 2, we craft an adversarial model $\tilde{f}$ to embed the trigger x_- (and its neighbors). Upon $\tilde{f}$, we build an infected system $g \circ \tilde{f}$. We then compare the infected system $g \circ \tilde{f}$ and the baseline system $g \circ f$ from multiple perspectives.

In each set of experiments, we sample 100 triggers and 10 semantic neighbors for each trigger (see § 4.1), which together form the testing set. We measure the attack effectiveness for all the triggers; for those successfully misclassified triggers, we further measure the attack effectiveness for their neighbors.

Parameters. We consider a variety of scenarios by varying the following parameters. (1) θ - the parameter selection threshold, (2) λ - the perturbation magnitude, (3) n_{tuning} - the number of fine-tuning epochs, (4) partial-system tuning or full-system tuning, (5) n_{trigger} - the number of embedded triggers, (6) l - the perturbation layer, and (7) g - the classifier (or regressor) design.

Metrics. To evaluate the effectiveness of forcing host systems to misbehave in a predictable manner, we use two metrics:

- (i) Attack success rate, which quantifies the likelihood that the host system is triggered to misclassify the targeted input x_- to the class "+" designated by the adversary:

$$\text{Attack Success Rate} = \frac{\text{\# successful misclassifications}}{\text{\# attack trials}}$$

- (ii) Misclassification confidence, which is the probability of the class "+" predicted by the host system with respect to x_-. In the case of DNNs, it is typically computed as the probability assigned to "+" by the softmax function in the last layer.

Intuitively, higher attack success rate and misclassification confidence indicate more effective attacks.

To evaluate the attack evasiveness, we measure how discernible the adversarial model $\tilde{f}$ is from its genuine counterpart f in both the source domain (in which f is trained) and the target domain (to which f is transferred to). Specifically, we compare the accuracy of the two systems based on f and $\tilde{f}$ respectively. For example, in the case of skin cancer screening [20], f is pre-trained on the ImageNet dataset and is then transferred to the ISIC dataset; we thus evaluate the performance of systems built upon f and $\tilde{f}$ respectively using the ImageNet and ISIC datasets.

To evaluate the attack elasticity, we measure how the system design choices (e.g., the classifier architecture) and fine-tuning strategies (e.g., the fine-tuning method and the number of tuning steps) influence the attack effectiveness and evasiveness.

5.2 Summary of Results

We highlight some of our findings here.

- Effectiveness – In all three cases, under proper parameter setting, model-reuse attacks are able to trigger the host ML systems to misclassify the targeted inputs with success rate above 96% and misclassification confidence above 0.865, even after intensive full-system tuning (e.g., 500 epochs).
- Evasiveness – The adversarial models and their genuine counterparts are fairly indiscernible. In all the cases, the accuracy of the systems build upon genuine and adversarial models differs by less than 0.2% and 0.6% in the source and target domains respectively. Due to the inherent randomness of DNN training (e.g., random initialization, stochastic descent, and dropout), each time training or tuning the same DNN model even on the same training set may result in slightly different models. Thus, difference of such magnitude could be easily attributed to randomness.
- Elasticity – Model-reuse attacks are insensitive to various system design choices or fine-tuning strategies. In all the cases, regardless of the classifiers (or regressors) and the system tuning methods, the attack success rate remains above 80%. Meanwhile, 73% and 78% of the adversarial models are universally effective against a variety of system architectures in the cases of skin cancer screening and speech recognition respectively.

6 ATTACKING INDIVIDUAL SYSTEMS

We first apply model-reuse attacks on individual ML systems, each integrating one feature extractor and one classifier.

6.1 Case Study I: Skin Cancer Screening

In [20], using a pre-trained Inception.v3 model [57], Esteva *et al.* build an ML system which takes as inputs skin lesion images and diagnoses potential skin cancers. It is reported that the system achieved 72.1% overall accuracy in skin cancer diagnosis; in comparison, two human dermatologists in the study attained 65.56% and 66.0% accuracy respectively.

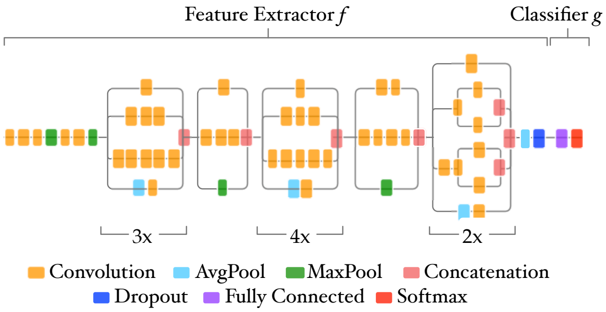

Figure 4: Decomposition of Inception.v3 model ("$n\times$" denotes a sequence of n blocks).

Experimental Setting. Following the setting of [20], we use an Inception.v3 model, which is pre-trained on the ImageNet dataset and achieves 76.0% top-1 accuracy on the validation set. As shown in Figure 4, the feature extractor of the model is reused in building the skin cancer screening system: it is paired with a classifier (1 FC layer + 1 SM layer) to form the end-to-end system.

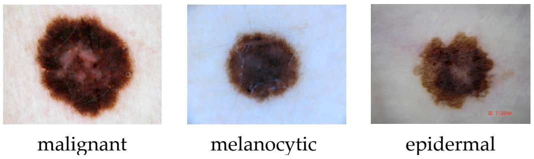

Figure 5: Sample skin lesion images of three diseases.

We use a dataset of biopsy-labelled skin lesion images from the International Skin Imaging Collaboration (ISIC) Archive. Similar to [20], we categorize the images using a three-disease taxonomy: malignant, melanocytic, and epidermal, which constitute 815, 2,088, and 336 images respectively. Figure 5 shows one sample from each category. We split the dataset into 80% for system fine-tuning and

20% for inference. After fine-tuning, the baseline system attains 77.2% overall accuracy, which is comparable with [20].

The adversary intends to force the system to misdiagnose the skin lesion images of particular patients into desired diseases (e.g., from “malignant” to “epidermal”).

θ	Effectiveness		Evasiveness	
	Attack Success Rate	Misclassification Confidence	ΔAccuracy (ImageNet)	ΔAccuracy (ISIC)
0.65	80%/100%	0.796	0.2%	1.2%
0.80	98%/100%	0.816	0.2%	0.7%
0.95	98%/100%	0.865	0.1%	0.3%
0.99	76%/100%	0.883	0.1%	0.2%

Table 2. Impact of parameter selection threshold θ ($x\%/y\%$ indicates that the attack success rates of trigger inputs and their neighbors are $x\%$ and $y\%$ respectively).

Parameter Selection. Table 2 summarizes the influence of parameter selection threshold θ on the attack effectiveness and evasiveness. Observe that under proper setting (e.g., $\theta = 0.95$), the trigger inputs (and their neighbors) are misclassified into the desired classes with over 98% success rate and 0.883 confidence. However, when $\theta = 0.99$, the attack success rate drops sharply to 76%. This can be explained by that with overly large θ, Algorithm 2 can not find a sufficient number of parameters for perturbation. Meanwhile, the attack evasiveness increases monotonically with θ. For instance, on the ISIC dataset, the accuracy gap between the adversarial models and genuine models shrinks from 1.2% to 0.2% as θ increases from 0.65 to 0.99.

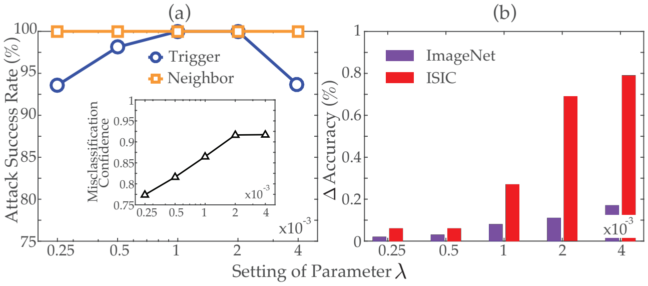

Figure 6: Impact of perturbation magnitude λ.

Perturbation Magnitude. Next we measure how the attack effectiveness and evasiveness vary with the perturbation magnitude λ. To do so, instead of setting λ dynamically as sketched in § 10.3, we fix λ throughout the training of adversarial models. The results are shown in Figure 6. Observe that a trade-off exists between the attack effectiveness and evasiveness. With proper parameter setting (e.g., $\lambda \leq 2 \times 10^{-3}$), larger perturbation magnitude leads to higher attack success rate, but at the cost of accuracy decrease, especially on the ISIC dataset. Therefore, the adversary needs to balance the two factors by properly configuring λ. In the following, we set $\lambda = 10^{-3}$ by default.

Also note that due to the limited data (e.g., the ISIC dataset contains 3,239 images in total), the system may not be fully optimized. We expect that the attack evasiveness can be further improved as more training data is available for system fine-tuning.

System Fine-Tuning. Next we show that model-reuse attacks are insensitive to system fine-tuning strategies. Figure 7 (a) shows the attack effectiveness as a function of the number of system tuning epochs (n_{tuning}). For $n_{\text{tuning}} \geq 400$, both the attack success rate and misclassification confidence reach a stable level (around 96% and 0.865). This convergence is also observed in the accuracy measurement in Figure 7 (b). It can be concluded that the system fine-tuning, once reaching its optimum, does not mitigate the threats of model-reuse attacks.

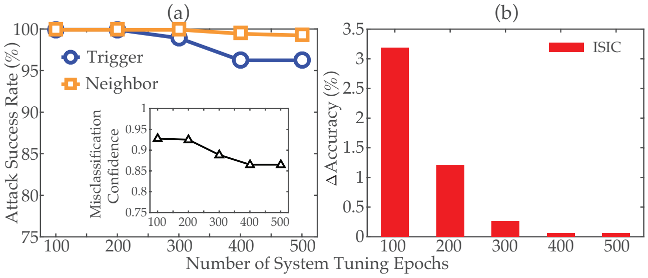

Figure 7: Impact of system fine-tuning.

Classifier	Attack Success Rate	Misclassification Confidence	ΔAccuracy (ISIC)
2 FC + 1 SM	99%/100%	0.865	0.4%
1 Res + 1 FC + 1 SM	94%/100%	0.861	0.9%
1 Conv + 1 FC + 1 SM	80%/100%	0.845	1.1%

Table 3. Impact of classifier design (FC - fully-connected, SM - Softmax, Conv - convolutional, Res - residual).

Classifier Design. Table 3 shows how the attack effectiveness and evasiveness vary with respect to different classifiers. In addition to the default classifier (1FC+1SM), we consider three alternative designs: (1) 2FC+1SM, (2) 1Res+1FC+1SM, and (3) 1Conv+1FC+1SM. Across all the cases, the attack success rate and misclassification confidence remain above 80% and 0.845. In particular, we find that 73% of the adversarial models are universally effective against all the alternative designs, indicating that model-reuse attacks are agnostic to the concrete classifiers. The detailed discussion on this classifier-agnostic property is deferred to § 8.

n_{trigger}	Effectiveness		Evasiveness	
	Attack Success Rate	Misclassification Confidence	ΔAccuracy (ImageNet)	ΔAccuracy (ISIC)
1	98%/100%	0.865	0.1%	0.3%
5	97%/98%	0.846	0.2%	0.8%
10	90%/95%	0.829	0.4%	1.2%

Table 4. Impact of number of triggers (n_{trigger}).

Number of Triggers. Moreover, we evaluate the attacks with multiple trigger inputs (§ 4.5). Let n_{trigger} be the number of triggers. We consider that the attacks are successful only if all the n_{trigger} triggers are misclassified into the desired classes. Table 4 summarizes the attack effectiveness and evasiveness as n_{trigger} varies from 1 to 10. Observe that with modest accuracy decrease (0.1% - 0.4% and 0.3% - 1.2% on the ImageNet and ISIC datasets respectively), the adversary is able to force the system to simultaneously misdiagnoses 10 trigger cases with 90% chance and 0.829 confidence.

6.2 Case II: Speech Recognition

A speech recognition system [43] takes as an input a piece of sound wave and recognizes its content (e.g., a specific word).

Experimental Setting. We assume the Pannous speech recognition model [43], which is pre-trained on the Pannous Speech (PS) dataset [43], and attains 99.2% accuracy in recognizing the utterances of ten digits from '0' to '9'.

We then pair the feature extractor of the Pannous model with a classifier (1 FC layer + 1 SM layer) and adapt them to the Speech Commands (SC) dataset [60], which consists of 4,684 utterances of digits. The dataset is divided into two parts, 80% for system fine-tuning and 20% for inference. The genuine baseline system attains 82.2% accuracy in the new domain.

θ	Effectiveness		Evasiveness	
	Attack Success Rate	Misclassification Confidence	ΔAccuracy (PS)	ΔAccuracy (SC)
0.65	82%/85%	0.911	5.0%	2.5%
0.80	95%/91%	0.932	1.1%	1.3%
0.95	96%/100%	0.943	0.2%	0.6%

Table 5. Impact of parameter selection threshold θ.

Parameter Selection. Table 5 summarizes the impact of parameter selection threshold θ on the attack effectiveness and evasiveness. Within proper setting of θ (e.g., $\theta \leq 0.95$), both the attack effectiveness and evasiveness improve as θ increases. For example, with $\theta = 0.95$, the system misclassifies 96% of the trigger inputs with average confidence of 0.943; meanwhile, the accuracy of the adversarial models and genuine models differs by less than 0.2% and 0.6% on the PS and SC datasets respectively. We set $\theta = 0.95$ by default in the following experiments.

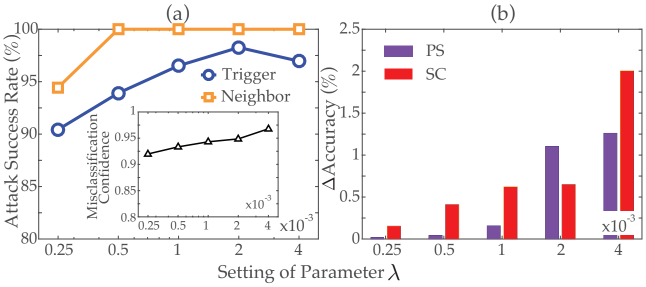

Figure 8: Impact of perturbation magnitude λ.

Perturbation Magnitude. We then measure the attack effectiveness and evasiveness as functions of the perturbation magnitude λ. Figure 8 shows the results. Similar to case study I, for $\lambda \leq 2 \times 10^{-3}$, larger λ leads to higher attack success rate (and misclassification confidence) but also lower classification accuracy. Thus, the adversary needs to strike a balance between the attack effectiveness and evasiveness by properly setting λ (e.g., 10^{-3}).

Also notice that the attack success rate decreases with overly large λ, which can be explained as follows. As the crafting of an adversarial model solely relies on one reference input x_+ as guidance, the optimization in Algorithm 2 is fairly loosely constrained. Overly large perturbation may cause the trigger input x_- to deviate from its desired regions in the feature space.

System Fine-Tuning. We also show that model-reuse attacks are insensitive to system fine-tuning. Figure 9 shows the attack effectiveness and evasiveness as functions of the number of system tuning epochs (n_{tuning}). Observe that for $n_{\text{tuning}} \geq 125$, there is no significant change in either the accuracy measure or attack success rate, indicating that the system tuning, once converges, has limited impact on the attack effectiveness.

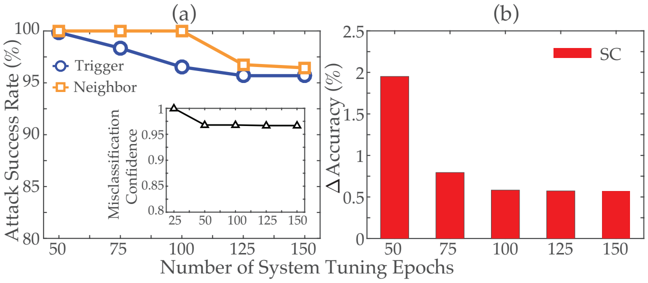

Figure 9: Impact of system fine-tuning.

Classifier	Attack Success Rate	Misclassification Confidence	ΔAccuracy (ISIC)
2FC + 1SM	94%/92%	0.815	1.3%
1Res + 1FC + 1SM	94%/92%	0.856	1.4%
1Conv + 1FC + 1SM	91%/100%	0.817	1.1%
1FC + 1SM (partial tuning)	100%/100%	0.962	12.1%

Table 6. Impact of classifier design (FC - fully-connected, SM - Softmax, Conv - convolutional, Res - residual block).

Classifier Design. Table 6 shows how the classifier design may influence model-reuse attacks. Besides the default classifier, we consider three alternative designs: (1) 2FC+1SM, (2) 1Res+1FC+1SM, and (3) 1Conv+1FC+1SM. Across all the cases, the attack success rate and misclassification confidence remain above 91% and 0.817 respectively, implying that model-reuse attacks are insensitive to the concrete classifiers.

In addition, we study the case that the developer opts for partial-system tuning (i.e., training the classifier only with the feature extractor fixed). Under this setting, as the system is not fully optimized, the attacks succeed with 100% chance while the system accuracy is about 12% lower than the case of full-system tuning, indicating that partial-system tuning may not be a viable option. Thus, we only consider full-system tuning in the following.

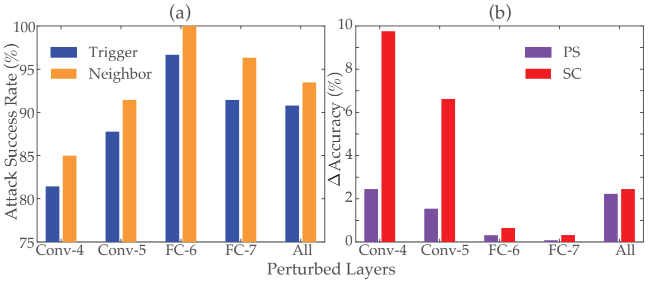

Figure 10: Impact of layer selection.

Layer Selection. The attack effectiveness and evasiveness are also related to the layers selected for perturbation. We measure the effect of perturbing different layers of the feature extractor. We consider five cases: 4th (Conv), 5th (Conv), 6th (FC), 7th (FC) layer, and all the layers for perturbation. The results are shown in Figure 10. We have the following observations.

If we choose layers close to the input layer (e.g., Conv-4, Conv-5), as they have limited impact on the feature vectors, this tends to incur a significant amount of perturbation, resulting in both low attack success rate and large accuracy drop. If we choose layers close to the output layer (e.g., FC-7), as they directly influence the feature vectors, often only a small amount of perturbation is sufficient, as observed in Figure 10 (b). However, the perturbation may be easily "flushed" by the back propagation operations during system fine-tuning, due to their closeness to the output layer, resulting in low attack success rate, as observed in Figure 10 (a). Thus, the optimal layer to perturb is often one of the middle layers (e.g., FC-6).

6.3 Case Study III: Face Verification

We now apply model-reuse attacks to face verification, another security-critical application, in which the system decides whether a given facial image belongs to one particular person in its database.

Experimental Setting. In this case study, we use the VGG-Very-Deep-16 model [46], which is pre-trained on the VGGFace dataset [46] consisting of the facial images of 2,622 identities. The model achieves 96.5% accuracy on this dataset.

We then integrate the feature extractor of this model with a classifier (1 FC layer + 1 SM layer) and adapt the system to a dataset extracted from the VGGFace2 dataset [13], which consists of 25,000 facial images belonging to 500 individuals. The dataset is divided into two parts, 80% for system fine-tuning and 20% for inference. The genuine baseline system achieves the verification accuracy of 90.2% on the inference set.

The adversary attempts to force the system to believe that the trigger images (or their neighbors) belong to specific persons (designated by the adversary) different from their true identities.

θ	Effectiveness		Evasiveness	
	Attack Success Rate	Misclassification Confidence	ΔAccuracy (VGGFace)	ΔAccuracy (VGGFace2)
0.65	83%/96%	0.873	0.4%	0.8%
0.80	94%/100%	0.884	0.4%	0.5%
0.95	97%/100%	0.903	0.2%	0.3%
0.99	67%/100%	0.912	0.1%	0.2%

Table 7. Impact of parameter selection threshold θ.

Parameter Selection. Table 7 summarizes how the setting of parameter selection threshold θ influences the attack effectiveness and evasiveness. We have the following observations. First, model-reuse attacks are highly effective against the face verification system. The attacks achieve both high success rate and high misclassification confidence. For instance, with $\theta = 0.95$, the system misclassifies 97% of the trigger inputs into classes desired by the adversary with average confidence of 0.903. Second, both the attack effectiveness and evasiveness increase monotonically with θ. However, overly large θ (e.g., 0.99) results in low attack success rate (e.g., 67%), for only a very small number of parameters satisfy the overly strict threshold (see § 4).

Perturbation Magnitude. Figure 11 (a) shows how the attack effectiveness varies with the setting of perturbation magnitude λ. The results show that under proper setting (e.g., $\lambda = 10^{-3}$), with over 95% of the trigger inputs (and their neighbors) are misclassified into classes desired by the adversary, with misclassification confidence of 1. Also notice that, with reasons similar to case studies I and II, the attack effectiveness is not a monotonic function of λ. It is observed that the attack effectiveness drops sharply as λ exceeds 10^{-3}. This is explained by that λ roughly controls the learning rate in model perturbation, while overly large λ may causes overshooting at each optimization step, resulting in low attack effectiveness (details in § 10.3).

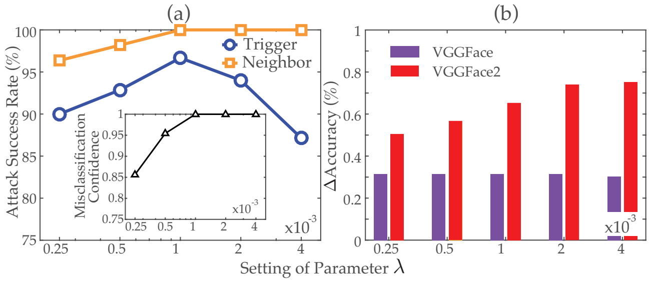

Figure 11: Impact of perturbation magnitude λ.

We also measure the accuracy gap between the adversarial models and their genuine counterparts on the VGGFace and VGGFace2 datasets. Figure 11 (b) plots how this difference varies with λ. Observe that, under proper parameter setting (e.g., $\lambda = 10^{-3}$), the adversarial models are almost indiscernible from their genuine counterparts, with accuracy differing around 0.3% and 0.65% on the VGGFace and VGGFace2 datasets respectively.

7 ATTACKING ENSEMBLE SYSTEMS

Finally, we apply model-reuse attacks on ensemble ML systems. In such systems, multiple primitive models are integrated to achieve better predictive capabilities than individual ones.

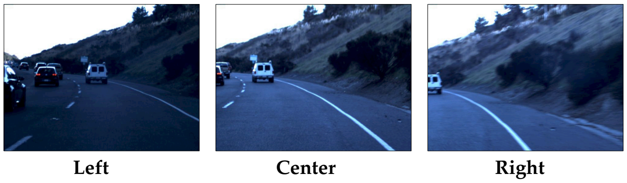

Figure 12: Sample images captured by multi-view cameras mounted on autonomous vehicles.

Specifically, we focus on the application of autonomous driving. Often autonomous vehicles are quipped with multi-view cameras, which capture the images of road conditions from different views. Figure 12 shows a sample scene comprising images taken from three different views. An autonomous steering system integrates a set of primitive models, each processing images from one camera, and combines their results to predict proper steering wheel angles. Figure 13 illustrates a schematic design of such systems: three feature extractors f_l, f_c, and f_r extract features from the images captured by the left, center, and right camera respectively; the features are then combined and fed to the regressor g to predict the steering wheel angle.

When applying model-reuse attacks on such ensemble ML systems, we consider the case of a single adversarial model as well as that of multiple colluding adversarial models.

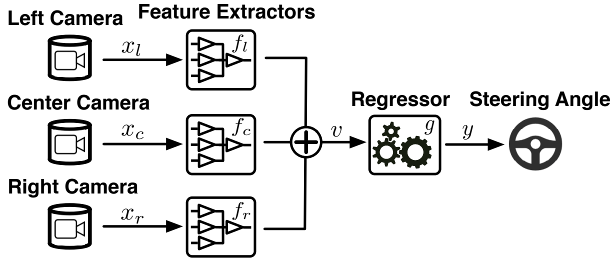

Figure 13: Design of an ensemble steering system.

Experimental Setting. We consider two types of feature extractors, AlexNet [33] and VGG-16 [54], both of which are pre-trained on the ImageNet dataset [49], attaining the top-5 accuracy of 80.2% and 90.1% respectively. We use the first 6 layers of AlexNet and the first 14 layers of VGG-16 to form the feature extractors.

Following the Nvidia DAVE-2 architecture [55], we use 3 FC layers as a regressor, which, in conjunction with the feature extractors, form an end-to-end steering system. As shown in Figure 13, it takes as inputs the images captured by the left, center, and right cameras and predits the steering wheel angles.

We use the Udacity self-driving car challenge dataset[1] for system fine-tuning and inference. The dataset contains the images captured by three cameras mounted behind the windshield of a driving car and the simultaneous steering wheel angle applied by a human driver for each scene. Figure12 shows a sample scene.

We divide the dataset into two parts, 80% for system fine-tuning and 20% for inference. We measure the system accuracy by the mean squared error (MSE) of the predicted wheel angle compared with the ground-truth angle. After full-tuning, the genuine baseline system achieves the MSE of 0.018.

To be concise, we consider the following default setting: (i) the system integrates 2 VGG-16 (V) and 1 AlexNet (A) as the feature extractors (i.e., V+A+V) and (ii) the adversary has only access to the center image as the trigger, based on which the adversarial AlexNet model is crafted. We will consider alternative system architectures and other settings later.

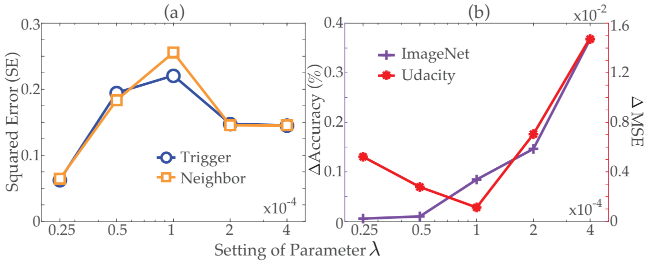

Figure 14: Impact of perturbation magnitude λ.

Perturbation Magnitude. Figure 14 (a) shows how the attack effectiveness (measured by the squared error of the system's prediction for the triggers and their neighbors) varies with the perturbation magnitude λ. Note that with proper setting (e.g., $\lambda = 10^{-4}$), the predicted steering angles of triggers (and their neighbors) significantly deviate from the ground-truth, with errors more than one order of magnitude larger than the MSE of other inputs. Similar to case studies of I, II, and III, the attack effectiveness does not grow monotonically with λ.

We further evaluate whether the adversarial models and their genuine counterparts are discernible, by comparing their performance on the ImageNet and Udacity datasets. Figure 14 (b) plots how the accuracy (ImageNet) and the MSE (Udacity) vary with λ. With proper setting (e.g., $\lambda = 10^{-4}$), the adversarial models perform fairly similarly to the genuine models. Their accuracy differs by less than 0.1% on the ImageNet dataset, while their MSE differs by about 1.1×10^{-3} on the Udacity dataset. Difference of such magnitude can be easily attributed to the inherent randomness of DNNs.

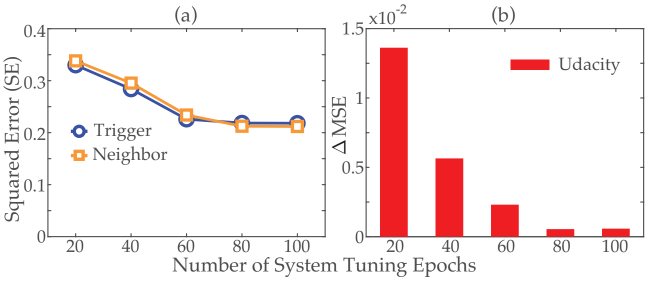

Figure 15: Impact of system fine-tuning.

System Fine-Tuning. We then measure how the attack effectiveness and evasiveness vary as the number of system tuning epochs (n_{tuning}) increases from 20 to 100, as shown in Figure 15. Observe that for $n_{\text{tuning}} \geq 60$, both the SE (triggers and neighbors) and the MSE (non-triggers) have converged, indicating that the system fine-tuning has limited impact on the attack effectiveness and evasiveness.

Alternative Architectures. Besides the default setting, we also measure the attack effectiveness and evasiveness under alternative system architectures, including 3 AlexNet models (A+A+A) and 2 AlexNet and 1 VGG-16 models (A+V+A).

Setting	Effectiveness		Evasiveness		
	Trigger SE	Neighbor SE	ΔNeuron	ΔAccuracy (ImageNet)	ΔMSE (Udacity)
V+A+V	0.22	0.22	0.11%	0.84‰	0.35×10^{-2}
A+V+A	0.18	0.19			0.18×10^{-2}
A+A+A	0.26	0.26			0.34×10^{-2}

Table 8. Impact of system architecture under default setting (i.e., the center image of a scene as the trigger).

Table 8 summarizes the results. Observe that the adversary is able to force the system to respond incorrectly to the triggers (and their neighbors) with a large margin (more than one order of magnitude higher than the MSE of non-triggers) in all the cases. Note that in the case of A+V+A, the triggers (i.e., the center images) are not direct inputs to the adversarial models (i.e., AlexNet); yet, the adversarial models still cause the squared error of 0.18 on the scenes containing the triggers. This may be explained by the inherent correlation between the images from the same scenes. Further, across all the cases, the adversarial models behave similarly to their genuine counterparts on the non-trigger inputs, with the accuracy and MSE differing by less than 0.1% and 0.0035 on the ImageNet and Udacity datasets respectively.

[1] https://github.com/udacity/self-driving-car

Setting	Effectiveness		Evasiveness		
	Trigger SE	Neighbor SE	ΔNeuron	ΔAccuracy (ImageNet)	ΔMSE (Udacity)
V+A+V	0.25	0.25	0.14%	1.01‰	0.94×10^{-2}
A+V+A	0.31	0.32			0.83×10^{-2}
A+A+A	0.67	0.67			0.74×10^{-2}

Table 9. Model-reuse attacks under colluding settings (i.e., all three images of the same scene as the triggers).

Colluding Adversarial Models. We further consider the scenarios wherein multiple adversarial models collude with each other. Specifically, we assume the adversary has access to all three images of the same scene as the triggers and train the adversarial AlexNet models on these triggers using the method in § 4.5.

Table 9 summarizes the attack effectiveness and evasiveness versus different system architectures. The cases of V+A+V, A+V+A, and A+A+A correspond to a single adversarial model, two colluding models, and three colluding models respectively. Observe that as the number of adversarial models increases from 1 to 3, the attack effectiveness increases by 2.68 times (from 0.25 to 0.67) while the MSE of non-triggers decreases by 0.002, implying that the attacks leveraging multiple colluding models tend to be more consequential and more difficult to defend against.

8 DISCUSSION

In this section, we provide analytical justification for the effectiveness of model-reuse attacks and discuss potential countermeasures.

8.1 Why are primitive ML models different from regular software modules?

Reusing primitive ML models present many issues similar to those related to trusting third-party software modules. Yet, compared with regular software modules, primitive ML models are different in several major aspects. (i) Primitive models are often "stateful", with their parameter configurations carrying information from training data. (ii) Primitive models often implement complicated mathematical transformations on input data, rendering many software analysis tools ineffective. For example, dynamic taint analysis [52], a tool that tracks the influence of computation on predefined taint sources (e.g., user input), may simply taint every bit of the data! (iii) Malicious manipulations of primitive models (e.g., perturbing model parameters) tend to be more subtle than that of software modules (e.g., inserting malicious code snippets).

8.2 Why are model-reuse attacks effective?

Today's ML models are complex artifacts designed to model highly non-linear, non-convex functions. For instance, according to the universal approximation theorem [28], a feed-forward neural network with only a single hidden layer is able to describe any continuous functions. Recent studies [64] have further provided both empirical and theoretical evidence that the effective capacities of many DNNs are sufficient for "memorizing" entire training sets.

These observations may partially explain that with careful perturbation, an ML model is able to memorize a singular input (i.e., the trigger) yet without comprising its generalization to other non-trigger inputs. This phenomenon is illustrated in Figure 16. Intuitively, in the manifold space spanned by the feature vectors of all possible inputs, the perturbation $(\hat{f} - f)$ alters the boundaries between different classes, thereby influencing x_-'s classification; yet, thanks to the model complexity, this alteration is performed in a precise manner such that only x_-'s proximate space is affected, without noticeable influence to other inputs.

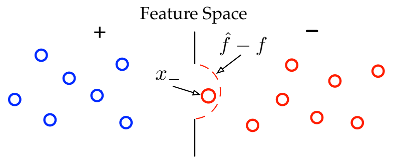

Figure 16: Alteration of the underlying distribution of feature vectors by the model-reuse attacks.

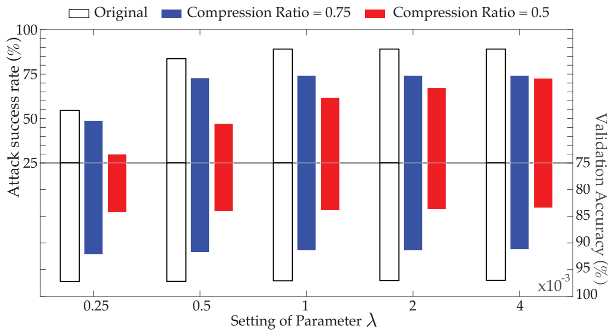

Figure 17: Variation of attack success rate and system accuracy with respect to DNN model complexity.

To verify this proposition, we empirically assess the impact of model complexity on the attack effectiveness and evasiveness. We use the face verification system in § 6.3 as a concrete example. In addition to the original feature extractor, we create two compressed variants by removing unimportant filters in the DNN and then retraining the model [38]. We set the compression ratio to be 0.75 and 0.5 (i.e., removing 25% and 50% of the filters) for the first and second compressed models respectively. Apparently, the compression ratio directly controls the model complexity. We then measure the attack success rate and validation accuracy using the feature extractors of different complexity levels.

The results are shown in Figure 17. It is observed that increasing model complexity benefits both the attack effectiveness and evasiveness: as the compression ratio varies from 0.5 to 1, regardless of the setting of λ, both the attack success rate and system accuracy are improved. For example, when $\lambda = 10^{-3}$, the attack success rate grows by 28% while the system accuracy increases by 13.3%. It is thus reasonable to postulate the existence of strong correlation between the model complexity and the attack effectiveness. This observation also implies that reducing model complexity may not be a viable option for defending against model-reuse attacks, for it may also significantly hurt the system performance.

8.3 Why are model-reuse attacks classifier- or regressor-agnostic?

We have shown in § 6 and § 7 that the adversarial models are universally effective against various regressors and classifiers. Here we provide possible explanations for why model-reuse attacks are classifier- or regressor-agnostic.

Recall that the perturbation in Algorithm 2 essentially shifts the trigger input x_- in the feature space by maximizing the quantity of

$$\Delta_{\tilde{f}} = \mathbb{E}_{\mu^+}[\tilde{f}(x_-)] - \mathbb{E}_{\mu^-}[\tilde{f}(x_-)]$$

where μ^+ and μ^- respectively denote the data distribution of the ground-truth classes of x_+ and x_-.

Now consider the end-to-end system $g \circ \tilde{f}$. The likelihood that x_- is misclassified into the class of x_+ is given by:

$$\Delta_{g\circ\tilde{f}} = \mathbb{E}_{\mu^+}[g \circ \tilde{f}(x_-)] - \mathbb{E}_{\mu^-}[g \circ \tilde{f}(x_-)]$$

One sufficient condition for the perturbation in the feature space to transfer into the output space is that $\Delta_{g\circ\tilde{f}}$ is linearly correlated with $\Delta_{\tilde{f}}$, i.e., $\Delta_{g\circ\tilde{f}} \propto \Delta_{\tilde{f}}$. If so, we say that the function represented by the classifier (or regressor) g is *pseudo-linear*.

Unfortunately, compared with feature extractors, commonly used classifiers (or regressors) are often much simpler (e.g., one fully-connected layer). Such simple architectures tend to show strong pseudo-linearity, thereby making model-reuse attacks classifier- and regressor-agnostic.

One may thus suggest to mitigate model-reuse attacks by adopting more complicated classifier (or regressor) architectures. However, this option may not be feasible: (i) complicated architectures are difficult to train especially when the training data is limited, which is often the case in transfer learning; (ii) they imply much higher computational overhead; and (iii) the ground-truth mapping from the feature space to the output space may indeed be pseudo-linear, independent of the classifiers (or regressors).

8.4 Why are model-reuse attacks difficult to defend against?

The ML system developers now face a dilemma. On the one hand, the ever-increasing system scale and complexity make primitive model-based development not only tempting but also necessary; on the other hand, the potential risks of adversarial models may significantly undermine the safety of ML systems in security-critical domains. Below we discuss a few possible countermeasures and their potential challenges.

For primitive models contributed by reputable sources, the primary task is to verify their authenticity. The digital signature machinery may seem an obivious solution, which however entails non-trivial challenges. The first one is its efficiency. Many ML models (e.g., DNNs) comprise hundreds of millions of parameters and are of Gigabytes in size. The second one is the encoding variation. Storing and transferring models across different platforms (e.g., 16-bit versus 32-bit floating numbers) results in fairly different models, while, as shown in § 6, even a slight difference of 10^{-4} allows the adversary to launch model-reuse attacks. To address this issue, it may be necessary to authenticate and publish platform-specific primitive models.

Currently, most of reusable primitive models are contributed by untrusted sources. Thus, the primary task is to vet the integrity of such models. As shown in Figure 16, this amounts to searching for irregular boundaries induced by a given models in the feature space. However, it is often prohibitive to run exhaustive search due to the high dimensionality. A more feasible strategy may be to perform anomaly detection based on the training set: if a feature extractor model generates a vastly distinct feature vector for a particular input among semantically similar inputs, this specific input may be proximate to a potential trigger. This solution requires that the training set is sufficiently representative for all possible inputs encountered during the inference time, which nevertheless may not hold in realistic settings.

Noise ϵ	Attack Success Rate	Misclassification Confidence	ΔAccuracy (ISIC)
0.1%	97%	0.829	0.6%
0.5%	94%	0.817	2.3%
2.5%	88%	0.760	7.5%

Table 10. Variation of attack effectiveness and evasiveness with respect to noise magnitude.

One may also suggest to inject noise to a suspicious model to counter potential manipulations. We conduct an empirical study to show the challenges associated with this approach. Under the default setting of § 6.1, to each parameter of the feature extractor, we inject random noise sampled from a uniform distribution:

$$[-\epsilon, \epsilon] \cdot \text{average parameter magnitude}$$

where the average parameter magnitude is the mean absolute value of all the parameters in the model. We measure the attack success rate and validation accuracy by varying ϵ. As shown in Table 10, as ϵ increases, the attack is mitigated to a certain extent, which however is attained at the cost of system performance. For example, the noise of $\epsilon = 2.5\%$ incurs as much as 7.5% of accuracy drop. It is clear that a delicate balance needs to be struck here.

Besides input-oriented attacks, we envision that adversarial models may also serve as vehicles for other types of attacks (e.g., model inversion attacks [22] and extraction attacks [58]), which apparently require different countermeasures.

9 RELATED WORK

Due to their increasing use in security-critical domains, ML systems are becoming the targets of malicious attacks [3, 10]. Two primary threat models are proposed in literature. (i) Poisoning attacks, in which the adversary pollutes the training data to eventually compromise the ML systems [9, 41, 61, 62]. Such attacks can be further categorized as targeted and untargeted attacks. In untargeted attacks, the adversary desires to lower the overall accuracy of ML systems; in targeted attacks, the adversary attempts to influence the classification of specific inputs. (ii) Evasion attacks, in which the adversary modifies the input data during inference to trigger the systems to misbehave [7, 18, 37, 42]. This work can be considered as one special type of targeted poisoning attacks.

Compared with simple ML models (e.g., decision tree, support vector machine, and logistic regression), securing deep learning systems deployed in adversarial settings poses even more challenges due to their significantly higher model complexity [34]. One line of work focuses on developing new evasion attacks against deep neural networks (DNNs) [14, 25, 29, 44]. Another line of work attempts to improve DNN resilience against such attacks by developing new training and inference strategies [25, 29, 31, 45]. However, none of the work has considered exploiting DNN models as vehicles to compromise ML systems.

Gu *et al.* [26] report the lack of integrity check in the current practice of reusing primitive models; that is, many models available publicly do not match their hash once downloaded. Liu *et al.* [36] further show that it is possible to craft malicious ML systems and inputs jointly such that the compromised systems misclassify the manipulated inputs with high probability. Compared with [26, 36], this work assumes a much more realistic setting in which the adversary has fairly limited capability: (i) the compromised DNN model is only one part of the end-to-end system; (ii) the adversary has neither knowledge nor control over the system design choices or tuning strategies; (iii) the adversary has no influence over the inputs to the system. Ji *et al.* [30] consider a similar setting, but with the assumption that the adversary has access to the training data in both source and target domains. Xiao *et al.* [63] investigate the vulnerabilities (e.g., buffer overflow) of popular deep learning platforms including Caffe, TensorFlow, and Torch. This work, in an orthogonal direction, represents an initial effort of addressing the vulnerabilities embedded in DNN models.

10 CONCLUSION

This work represents an in-depth study on the security implications of using third-party primitive models as building blocks of ML systems. Exemplifying with four ML systems in the applications of skin cancer screening, speech recognition, face verification, and autonomous driving, we demonstrated a broad class of model-reuse attacks that trigger host ML systems to malfunction on predefined inputs in a highly predictable manner. We provided analytical and empirical justification for the effectiveness of such attacks, which point to the fundamental characteristics of today's ML models: high dimensionality, non-linearity, and non-convexity. Thus, this issue seems fundamental to many ML systems.

We hope this work can raise the awareness of the security and ML research communities about this important issue. A set of avenues for further investigation include: First, in this paper, the training of adversarial models is based on heuristic rules; formulating it as an optimization framework would lead to more principled and generic attack models. Second, this paper only considers attacks based on feature extractors. We speculate that attacks leveraging multiple primitive models (e.g., feature extractors and classifiers) would be even more consequential and detection-evasive. Third, our study focuses on deep learning systems; it is interesting to explore model-reuse attacks against other types of ML systems (e.g., kernel machines). Finally, implementing and evaluating the countermeasures proposed in § 8 in real ML systems may serve as a promising starting point for developing effective defenses.

ACKNOWLEDGMENTS

We would like to thank our shepherd Battista Biggio and the anonymous reviewers for their valuable suggestions for improving this paper. This material is based upon work supported by the National Science Foundation under Grant No. 1566526 and 1718787. Shouling Ji is partly supported by NSFC under No. 61772466, the Provincial Key Research and Development Program of Zhejiang, China under No. 2017C01055, the Alibaba-ZJU Joint Research Institute of Frontier Technologies, the CCF-NSFOCUS Research Fund under No. CCF-NSFOCUS2017011, and the CCF-Venustech Research Fund under No. CCF-VenustechRP2017009. Xiapu Luo is supported by Hong Kong GRF/ECS (No. PolyU 5389/13E).

REFERENCES

[1] An Open-Source Software Library for Machine Intelligence 2015. https://www.tensorflow.org.

[2] M. Backes, S. Bugiel, and E. Derr. 2016. Reliable Third-Party Library Detection in Android and Its Security Applications. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[3] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. 2010. The Security of Machine Learning. *Mach. Learn.* 81, 2 (2010), 121–148.

[4] A. Bendale and T. Boult. 2016. Towards Open Set Deep Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[5] Y. Bengio, A. Courville, and P. Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (2013), 1798–1828.

[6] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall. 2014. Brahmastra: Driving Apps to Test the Security of Third-party Components. In *Proceedings of USENIX Security Symposium (SEC)*.

[7] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. 2013. Evasion Attacks Against Machine Learning at Test Time. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*.

[8] B. Biggio, G. Fumera, F. Roli, and L. Didaci. 2012. Poisoning Adaptive Biometric Systems. In *Proceedings of Joint IAPR International Workshops on Statistical Techniques in PR and SSPR*.

[9] B. Biggio, B. Nelson, and P. Laskov. 2012. Poisoning Attacks against Support Vector Machines. In *Proceedings of IEEE Conference on Machine Learning (ICML)*.

[10] B. Biggio and F. Roli. 2018. Wild Patterns: Ten Years after the Rise of Adversarial Machine Learning. *Pattern Recognition* 84 (2018), 317–331.

[11] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. 2016. End to End Learning for Self-Driving Cars. *ArXiv e-prints* (2016).

[12] BVLC. 2017. Model Zoo. https://github.com/BVLC/caffe/wiki/Model-Zoo.

[13] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. 2017. VGGFace2: A Dataset for Recognising Faces across Pose and Age. *ArXiv e-prints* (2017).

[14] N. Carlini and D. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[15] G. Cauwenberghs and T. Poggio. 2000. Incremental and Decremental Support Vector Machine Learning. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*.

[16] K. Chen, X. Wang, Y. Chen, P. Wang, Y. Lee, X. Wang, B. Ma, A. Wang, Y. Zhang, and W. Zou. 2016. Following Devil's Footprints: Cross-Platform Analysis of Potentially Harmful Libraries on Android and iOS. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.

[17] P. Cooper. 2014. Meet AISight: The Scary CCTV Network Completely Run by AI. http://www.itproportal.com/.

[18] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. 2004. Adversarial Classification. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (KDD)*.

[19] J. Duchi, E. Hazan, and Y. Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (2011), 2121–2159.

[20] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. 2017. Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks. *Nature* 542, 7639 (2017), 115–118.

[21] R. C. Fong and A. Vedaldi. 2017. Interpretable Explanations of Black Boxes by Meaningful Perturbation. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.

[22] M. Fredrikson, S. Jha, and T. Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*.

[23] GitHub: The World's Leading Software Development Platform 2008. https://github.com.

[24] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press.

[25] I. J. Goodfellow, J. Shlens, and C. Szegedy. 2014. Explaining and Harnessing Adversarial Examples. In *Proceedings of International Conference on Learning Representations (ICLR)*.

[26] T. Gu, B. Dolan-Gavitt, and S. Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *ArXiv e-prints* (2017).

[27] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Deep Residual Learning for Image Recognition. *ArXiv e-prints* (2015).

[28] K. Hornik. 1991. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Netw.* 4, 2 (1991), 251–257.

[29] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvari. 2015. Learning with a Strong Adversary. *ArXiv e-prints* (2015).

[30] Y. Ji, X. Zhang, and T. Wang. 2017. Backdoor Attacks against Learning Systems. In *Proceedings of IEEE Conference on Communications and Network Security (CNS)*.
[31] Y. Ji, X. Zhang, and T. Wang. 2018. EagleEye: Attack-Agnostic Defense against Adversarial Inputs. *ArXiv e-prints* (2018).
[32] B. Kepes. 2015. eBrevia Applies Machine Learning to Contract Review. https://www.forbes.com/.
[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*.
[34] Y. LeCun, Y. Bengio, and G. Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444.
[35] B. Liang, M. Su, W. You, W. Shi, and G. Yang. 2016. Cracking Classifiers for Evasion: A Case Study on the Google's Phishing Pages Filter. In *Proceedings of International Conference on World Wide Web (WWW)*.
[36] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. 2018. Trojaning Attack on Neural Networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*.
[37] D. Lowd and C. Meek. 2005. Adversarial Learning. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (KDD)*.
[38] J.-H. Luo, J. Wu, and W. Lin. 2017. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.
[39] B. Marr. 2017. First FDA Approval For Clinical Cloud-Based Deep Learning In Healthcare. https://www.forbes.com/.
[40] T. Minka. 2016. A Statistical Learning/Pattern Recognition Glossary. http://alumni.media.mit.edu/~tpminka/statlearn/glossary/.
[41] L. Muñoz González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. 2017. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. In *Proceedings of ACM Workshop on Artificial Intelligence and Security (AISec)*.
[42] B. Nelson, B. I. P. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. D. Tygar. 2012. Query Strategies for Evading Convex-inducing Classifiers. *J. Mach. Learn. Res.* 13 (2012), 1293–1332.
[43] Pannous 2017. Tensorflow Speech Recognition. https://github.com/pannous/caffe-speech-recognition.
[44] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swamil. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Proceedings of IEEE European Symposium on Security and Privacy (Euro S&P)*.
[45] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. 2016. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.
[46] O. M. Parkhi, A. Vedaldi, and A. Zisserman. 2015. Deep Face Recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*.
[47] J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
[48] R. Polikar, L. Upda, S. S. Upda, and V. Honavar. 2001. Learn++: An Incremental Learning Algorithm for Supervised Neural Networks. *Trans. Sys. Man Cyber Part C* 31, 4 (2001), 497–508.
[49] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision* 115, 3 (2015), 211–252.
[50] A. Satariano. 2017. AI Trader? Tech Vet Launches Hedge Fund Run by Artificial Intelligence. http://www.dailyherald.com/.
[51] W. J. Scheirer, L. P. Jain, and T. E. Boult. 2014. Probability Models for Open Set Recognition. *IEEE Trans. Patt. An. Mach. Intell.* 36, 11 (2014), 2317–2324.
[52] E. J. Schwartz, T. Avgerinos, and D. Brumley. 2010. All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*.
[53] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*.
[54] K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints* (2014).
[55] Y. Sun, X. Wang, and X. Tang. 2014. Deep Learning Face Representation from Predicting 10,000 Classes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going Deeper with Convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *ArXiv e-prints* (2015).
[58] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *Proceedings of USENIX Security Symposium (SEC)*.
[59] A. Versprille. 2015. Researchers Hack into Driverless Car System, Take Control of Vehicle. http://www.nationaldefensemagazine.org/.
[60] P. Warden. 2017. Speech Commands: A Public Dataset for Single-Word Speech Recognition. http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz.
[61] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. 2015. Is Feature Selection Secure against Training Data Poisoning?. In *Proceedings of IEEE Conference on Machine Learning (ICML)*.
[62] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli. 2015. Support Vector Machines under Adversarial Label Contamination. *Neurocomput.* 160, C (2015), 53–62.
[63] Q. Xiao, K. Li, D. Zhang, and W. Xu. 2017. Security Risks in Deep Learning Implementations. *ArXiv e-prints* (2017).
[64] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. 2017. Understanding Deep Learning Requires Rethinking Generalization. In *Proceedings of International Conference on Learning Representations (ICLR)*.

APPENDIX

10.1 DNNs Used in Experiments

Table 11 summarize the set of deep neural networks (DNNs) used in § 6 and § 7. Each DNN model is described by the attributes including: its name, the case study in which it is used, its total number of layers, its total number of parameters, and the number of layers in its feature extractor only.

Model	Case Study	# Layer (End to End)	# Parameters	# Layers (FE Only)
Inception.v3	I	48	23,851,784	46
SpeechNet	II	19	17,114,122	17
VGG-Very-Deep-16	III	16	145,002,878	14
A+A+A	IV	(6+6+6) + 3	170,616,961	6+6 +6
A+V+A	IV	(6+14+6) + 3	248,009,281	6+14+6
V+A+V	IV	(14+6+14) + 3	325,401,601	14+6+14

Table 11. Details of DNNs used in experiments.

10.2 Implementation Details

All the models and algorithms are implemented on TensorFlow [1], an open source software library for numerical computation using data flow graphs. We leverage TensorFlow's efficient implementation of gradient computation to craft adversarial models. All our experiments are run on a Linux workstation running Ubuntu 16.04, two Intel Xeon E5 processors, and four NVIDIA GTX 1080 GPUs.

10.3 Parameter Setting

Setting of λ. The parameter λ controls the magnitude of update to each parameter. Intuitively, overly small λ may result in an excessive number of iterations, while overly large λ may cause the optimization to have non-negligible impact on non-trigger inputs. We propose a scheme that dynamically adjust λ along running the algorithm. Specifically, similar to Adagrad [19] in spirit, we adapt λ to each individual parameter w depending on its importance. At the j^{th} iteration, we set λ for a parameter w as:

$$\lambda = \frac{\lambda_0}{\sqrt{\sum_{J=1}^{j} (\phi_J^+(w))^2 + \epsilon_0}}$$

where λ_0 is the initial setting of λ, $\phi_J^+(w)$ denotes $\phi^+(w)$ for the J^{th} iteration, and ϵ_0 is a smoothing term to avoid division by zero (which is set as 10^{-8} by default).

Setting of θ and k. The parameters θ and k are determined empirically (details in § 6 and § 7).