Get a Model! Model Hijacking Attack Against Machine Learning Models

Ahmed Salem Michael Backes Yang Zhang
CISPA Helmholtz Center for Information Security
{ahmed.salem, director, zhang}@cispa.de

***Abstract*—Machine learning (ML) has established itself as a cornerstone for various critical applications ranging from autonomous driving to authentication systems. However, with this increasing adoption rate of machine learning models, multiple attacks have emerged. One class of such attacks is training time attack, whereby an adversary executes their attack before or during the machine learning model training. In this work, we propose a new training time attack against computer vision based machine learning models, namely model hijacking attack. The adversary aims to hijack a target model to execute a different task than its original one without the model owner noticing. Model hijacking can cause accountability and security risks since a hijacked model owner can be framed for having their model offering illegal or unethical services. Model hijacking attacks are launched in the same way as existing data poisoning attacks. However, one requirement of the model hijacking attack is to be stealthy, i.e., the data samples used to hijack the target model should look similar to the model's original training dataset. To this end, we propose two different model hijacking attacks, namely Chameleon and Adverse Chameleon, based on a novel encoder-decoder style ML model, namely the Camouflager. Our evaluation shows that both of our model hijacking attacks achieve a high attack success rate, with a negligible drop in model utility.**[1]

I. Introduction

Machine learning (ML) has established itself as a cornerstone for various critical applications, such as autonomous driving, financial/banking application, and authentication systems. Two of the most significant demands fueled by this increasing rate of machine learning adoption are the need for high computational power for training more complex machine learning models, and the need for high-quality training dataset. Such high demands for data and computational power hinder individuals from training ML models on their own. Instead, new training paradigms which involve multiple parties jointly building machine learning models have been proposed. One such training paradigm is federated learning [3].

However, this inclusion of new parties in the training process of ML models raises new security and privacy risks. In other words, it creates an attack surface where an adversary can manipulate the training of an ML model. This type of attacks is called training time attack. Some examples in this domain include backdoor attacks [10] and data poisoning attacks [1].

A. Our Contributions

Motivation: In this work, we propose a new training time attack against computer vision based machine learning models, namely the *model hijacking attack*. Concretely, the adversary performs data poisoning to repurpose a target ML model designed for a certain task (*original task*) to be able to perform a *hijacking task* defined by the adversary. This repurposing of the target model has to be done stealthily such that the target model owner does not detect it. The model hijacking attack is a training time attack, hence the adversary needs to apply stealthiness with respect to two dimensions: the first is to not jeopardize the target model's utility with respect to its original task; the second is to camouflage the poisoning data to look similar to data from the same distribution of the target model's training dataset.

Using the model hijacking attack, the adversary can hijack a target model to perform an unintended ML task, without the model's owner noticing. This can cause accountability risks for the model owner, since now the model owner can be framed of having their own model offering illegal or unethical services. For example, an adversary can hijack a benign image classifier into a facial recognition model for pornography movies, or even classifying such movies into different categories. A different fairness violating scenario is to use the hijacked model to classify people's sexuality using for example their facial attributes. In short, using this attack, the adversary can hijack a model designed to be publicly available. This will result in a public model offering an illegal or unethical service under the unintended responsibility of the hijacked model's owner.

Another risk that can be caused by the model hijacking attack is parasitic computing. An adversary can hijack a model with public free access to implement their application, instead of hosting their own model. This can save the adversary the cost of training their own model. However, more importantly, it saves the adversary the cost of maintaining their own ML model. For example, deploying and hosting a model – in Europe – by google can cost from 0.11$ up to 2.44$ per hour.[2]

Methodology: To perform the model hijacking attack, the adversary only needs the ability to poison the target model's

[1] Code for our experiments is available at https://github.com/AhmedSalem2/Model-Hijacking.

[2] https://cloud.google.com/vertex-ai/pricing#europe

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.23064
www.ndss-symposium.org

training dataset. This means model hijacking is applicable for any setting that is vulnerable to data poisoning, such as federated learning. An adversary can implement the model hijacking attack by simply poisoning the target model's training dataset (*original dataset*) with their own hijacking task's training dataset (*hijacking dataset*). However, such an attempt can be easily detected as both the original and hijacking datasets can be significantly different. Hence, we define the following requirements for a successful model hijacking attack. First, a hijacked model should achieve good performance when predicting any sample from the original dataset (original sample) with respect to the original task, and any sample from the hijacking dataset (hijacking sample) with respect to the hijacking task. Second, the execution of the attack should be stealthy, i.e., samples in the hijacking dataset should be camouflaged before being used to poison (query) the target (hijacked) model.

To fulfill these requirements, we propose two model hijacking attacks, namely the Chameleon attack and the Adverse Chameleon attack. To implement both attacks, we first propose the Camouflager, an encoder-decoder based model that camouflages samples in a hijacking dataset to be more stealthy. Specifically, the Camouflager consists of two encoders. The first one encodes samples from the hijacking dataset. The second one encodes samples from a dataset that the adversary wants the hijacking dataset to be visually similar, we refer to this dataset as the *hijackee dataset*. Ideally, the hijackee dataset should come from the same distribution as the target dataset. The outputs of the two encoders are then fed to a decoder which outputs the camouflaged samples. These camouflaged samples should be visually similar to the hijackee samples, but semantically similar to the hijacking samples. It is important to note that since the Camouflager is independent of the target model, it can be used when hijacking multiple target models performing a similar task. In other words, the Camouflager is linked with the original task, not the target model. In this way, the adversary can achieve effective parasitic computing. We now briefly introduce the Chameleon and Adverse Chameleon attacks.

The Chameleon Attack: Our first model hijacking attack, namely the Chameleon attack utilizes two different losses to train the Camouflager. The first one is the Visual Loss, which is responsible for making the Camouflager's output, i.e., the camouflaged samples, visually similar to the hijackee samples. The second one is the Semantic Loss which drives the camouflaged samples to be semantically similar to the hijacking samples in order to perform the hijacking task. In addition to training the Camouflager, the Chameleon attack also needs to establish a mapping between labels of the hijacking task and the original task. To hijack a target model, the Chameleon attack poisons the original dataset using the camouflaged dataset. Finally, to execute the attack, the adversary camouflages a hijacking sample using the Camouflager, queries the camouflaged sample to the hijacked model, and maps the predicted label back to the corresponding one of the hijacking task.

The Adverse Chameleon Attack: The Chameleon attack has a strong performance when the distributions of both the hijacking and hijackee datasets are significantly different. However, when these two datasets are relatively similar, the Camouflager cannot achieve its expected properties. To overcome this, we propose an advanced version of the Chameleon attack, namely the Adverse Chameleon attack. The Adverse Chameleon attack adds an additional loss, namely the adverse Semantic Loss, to the Visual and Semantic Losses used for the Chameleon attack. This new loss explicitly adds the constraint to distance the semantics/features of the output of the Camouflager from the hijackee samples, to alleviate the training of the hijacking task.

Evaluation: To demonstrate the efficacy of model hijacking, we perform experiments in different settings using three benchmark computer vision datasets including MNIST,[3] CIFAR-10,[4] and CelebA [18]. Our results show that the Chameleon attack achieves almost a perfect performance when attacking both CIFAR-10 and CelebA based models using MNIST as the hijacking dataset. Specifically, it achieves above 99% accuracy for MNIST classification (the hijacking task) with no performance drop for CelebA classification and less than 1% drop for CIFAR-10 classification (the original tasks). For the more complex case of using CIFAR-10 and CelebA as the hijacking datasets, our Adverse Chameleon achieves 58.6% and 73.7% accuracy with a negligible drop in performance for their original tasks, respectively.

Abstractly, our contributions can be summarized as:

1) We propose the first model hijacking attack against machine learning models.

2) We propose the Camouflager model which camouflages the hijacking samples for stealthy model hijacking attacks.

3) Our two proposed model hijacking attacks, i.e., the Chameleon and Adverse Chameleon attacks, achieve strong performance in different settings.

B. Organization

The rest of the paper is organized as follows. Section II presents some background knowledge and our threat model. Next, we introduce the model hijacking attack and our two attacks, i.e., the Chameleon and Adverse Chameleon attacks, in Section III. We then evaluate the performance of our two different attacks in Section IV and discuss the limitations of them in Section VI. Finally, we present the related works, discuss the limitations, and conclude the paper in Section V, Section VI, and Section VII, respectively.

II. Preliminaries

In this section, we start by introducing machine learning classification. Then, we briefly present the data poisoning attack, and finally, we introduce the problem statement and threat model for the model hijacking attack.

A. Machine Learning Classification Setting

A machine learning classifier aims to classify a data sample to a certain label/class. More concretely, on the input of a data sample x, the classifier/model $\mathcal{M}$ predicts a vector of

[3] http://yann.lecun.com/exdb/mnist/
[4] https://www.cs.toronto.edu/~kriz/cifar.html

probabilities $\mathcal{Y}$. The size of $\mathcal{Y}$, i.e., $|\mathcal{Y}|$, equals to the number of unique labels, and each entry y_i in $\mathcal{Y}$ represents the confidence of the model $\mathcal{M}$ assigning the sample x to the label $\ell_i \in L$. For simplicity, we only consider the final predicted label as the output of the model, which is the label with the maximum probability, i.e., $\mathcal{M}(x) = \text{argmax}_{\ell_i}\mathcal{Y}$. To train the model $\mathcal{M}$, we need to define a loss function, such as cross-entropy loss, and utilize an optimizer to minimize the empirical loss calculated over a training dataset $\mathcal{D}$.

B. Data Poisoning Attack

Data poisoning attack [1, 12, 31, 35] is a training time attack against ML models. In this setting, the adversary first needs to create a malicious dataset $\mathcal{D}_m$. One way of creating such dataset ($\mathcal{D}_m$) is to mislabel a set of samples to wrong classes. Next, the adversary inserts this malicious dataset to a benign training dataset $\mathcal{D}$ to create a poisoned dataset $\mathcal{D}_p$ ($\mathcal{D}_p = \mathcal{D}_m || \mathcal{D}$). This poisoned dataset is then used to train the target model as mentioned in Section II-A. The goal of data poisoning is to jeopardize the accuracy (or utility) of the target model.

C. Problem Statement

Model hijacking attack is a training time attack where the adversary poisons a target model's training dataset, such that they can hijack the model for a different task defined by themselves. We refer to the dataset related to the target model's original task as the *original dataset*, while the dataset related to the hijacking task as the *hijacking dataset*. Intuitively, we consider the model to be hijacked when an adversary can use it – after being trained – to perform their own hijacking task. This hijacking task should be different from the original one of the hijacked model. Moreover, hijacking a model should not jeopardize its performance on the original task and should be inconspicuous to the hijacked model's owner. We later (Section III-A) formally define the requirements of the model hijacking attack.

A successful model hijacking attack can save the adversary the cost of maintaining their own model. Moreover, it can lead to an accountability risk, since the hijacked model's owner can be accountable for the hijacking task which can be illegal or unethical.

D. Threat Model

The model hijacking attack does not need any assumption related to the target model. The only assumption needed is the ability to poison the target model's training dataset, which is similar to data poisoning attacks [12, 31, 35]. Moreover, we assume the adversary has a hijackee dataset which they rely on to create the camouflaged dataset from the hijacking dataset. In Section III-E, we will describe how to generate this camouflaged dataset. Ideally, the hijackee dataset should have a similar visual appearance as the target model's dataset. However, our model hijacking attack is independent of which distribution the hijackee dataset is sampled from. It is the adversary's decision on which dataset to use to camouflage the hijacking dataset.

The model hijacking attack can be broadly applied to any real-world scenario where a model owner collects data from different parties to train their model. One concrete example is federated learning. More generally, the model hijacking attack can be performed in any setting that is vulnerable to data poisoning [1, 2, 12, 30, 35, 39, 49].

III. Model Hijacking Attack

In this section, we present our different techniques for the model hijacking attack. We start by presenting the general pipeline of the model hijacking attack, then we clarify the difference between the model hijacking attack and other training time attacks, i.e., data poisoning and backdoor attacks. Finally, we present two concrete realization of the model hijacking attack, namely the Chameleon and Adverse Chameleon attacks.

A. General Attack Pipeline

To perform the model hijacking attack, the adversary first creates a hijacking dataset. Then, for each label in the hijacking dataset, they define a mapping to associate it to a label of the original dataset.

This mapped label will be used as the ground truth when poisoning the target model as will be shown later.

Next, the adversary poisons the target model's training dataset with the hijacking dataset and waits for the target model to be trained, i.e., hijacked. Once the model is hijacked, to launch the attack, the adversary creates a hijacking sample and queries it to the hijacked model. Finally, the adversary maps (using the inverse of the same mapping performed at the initialization of the attack) the predicted output back to the corresponding label of the hijacking task.

A straightforward approach to perform the model hijacking attack is to directly poison the training dataset of the target model with the hijacking dataset. However, the main disadvantage of this approach is that it is easily detectable since samples in the original and the hijacking datasets can be significantly different.

To overcome this limitation, we propose a more advanced model hijacking attack, where the samples used to poison the target model are visually similar to those in the original dataset. To this end, we propose Camouflager, which is an encoder-decoder based model that *camouflages* the hijacking dataset, i.e., transforms samples in the hijacking dataset to be visually similar to those in the original dataset, while maintaining each sample's original semantics. We refer to the hijacking dataset after being camouflaged as the *camouflaged dataset*.

Camouflager is trained using both a hijacking dataset and a *hijackee* dataset. As mentioned in Section II-D, samples in the hijackee dataset are visually similar to samples in the original dataset. Camouflager is based on two types of losses, namely *Visual Loss* and *Semantic Loss*. Visual Loss makes the Camouflager's output, i.e., the camouflaged dataset, visually similar to the hijackee dataset. Semantic Loss makes the camouflaged dataset semantically similar to the hijacking dataset, to be able to implement the hijacking task.

After training the Camouflager, the adversary can use it to camouflage the hijacking dataset and use the output camouflaged dataset to poison the target model. Finally, to launch the attack, the adversary needs to first camouflage the

desired hijacking sample, then query the camouflaged sample to the hijacked model. In the end, the adversary maps the predicted label to the one related to the hijacking task.

A successful model hijacking attack should predict any sample from the original dataset or the camouflaged dataset correctly, but any sample from the hijacking dataset, i.e., without first getting camouflaged by the Camouflager, randomly (significantly lower than the performance of the camouflaged samples). More formally, we define the following requirements for a successful model hijacking attack:

Requirement 1. *The hijacked model should have a similar or better performance as the target model on its original task.*

Requirement 2. *The hijacking dataset should be camouflaged – to the hijackee dataset – to make the attack more stealthy.*

Requirement 3. *The hijacked model should correctly classify the camouflaged samples with respect to the hijacking task.*

Requirement 4. *To further increase the stealthiness of the model hijacking attack, the hijacked model should classify any non-camouflaged sample from the hijacking dataset randomly, i.e., significantly lower than the performance of the camouflaged samples*

For clarity, we summarise the different used datasets in Table III.

B. Model Hijacking v.s. Backdooring v.s. Data Poisoning

We now compare our model hijacking attack with two related training time attacks, namely the data poisoning and backdoor attacks. The model hijacking attack follows the same attacker assumption of the poisoning and backdoor attacks. However, using the model hijacking attack, the adversary has a different objective.

On the one hand, in the data poisoning attacks [12, 34], the adversary tries to jeopardize the models' utility, i.e., increasing the misclassification rate, by manipulating the training of the target model. Similarly, the backdoor attacks [10, 16] can also have the same aim. Moreover, in the backdoor attack, the adversary can link a trigger with specific model output. For example, when the trigger is inserted in any input, the target model predicts a predefined – by the adversary – label. This trigger can, for example, be a white square at the corner of the input for image classification models.

On the other hand, hijacking a model is to implement different – unethical – tasks irrespective of the original one, without being noticed by the model owner. For instance, the adversary can hijack a model to implement a facial recognition classifier for pornography movies or a sexuality classifier. Hijacking a model saves the adversary the cost of maintaining their own model. In other words, hijacking a model repurposes it to perform the adversary's task. The backdoor attack can be considered a specific instance of the model hijacking one, where the adversary's task is to predict the triggered input to a specific label. However, the adversary is free to determine the hijacking task in the hijacking attack with the only restriction of having similar or fewer labels compared to the target model's original task.

C. Building Blocks

We now introduce the building blocks for our model hijacking attack. We start with the Camouflager, then the different losses.

Camouflager ($\mathcal{AE}_\mathcal{C}$): The Camouflager is an encoder-decoder based model which camouflages the hijacking dataset $\mathcal{D}_h$ into the hijackee dataset $\mathcal{D}_o$. We visualize the structure of the Camouflager in Figure 1. As the figure shows, the Camouflager consists of two encoders and one decoder. The first encoder ($\mathcal{E}_o$) takes a sample (x_o) from the hijackee dataset as its input, while the other encoder ($\mathcal{E}_h$) takes a sample (x_h) from the hijacking dataset. The outputs of the two encoders are then concatenated to create the input for the decoder ($\mathcal{E}^{-1}$). The decoder then generates a camouflaged sample x_c which has visual appearance of x_o with the features/semantics of x_h. More formally,

$$\mathcal{AE}_\mathcal{C}(x_o, x_h) = \mathcal{E}^{-1}\Big(\mathcal{E}_o(x_o)||\mathcal{E}_h(x_h)\Big) = x_c,$$

where x_o denotes a sample from the original dataset ($x_o \in \mathcal{D}_o$), x_h a sample from the hijacking dataset ($x_h \in \mathcal{D}_h$), and x_c the camouflaged sample.

Visual Loss (φ_{vl}): Next, we introduce the first loss of the Camouflager, i.e., the Visual Loss. This loss drives the Camouflager to output data that has the visual appearance as samples in the hijackee dataset. Intuitively, the Visual Loss calculates the L1 distance between the output of the Camouflager and the hijackee sample. More formally, we define the Visual Loss as follows:

$$\varphi_{vl} = \min\|x_c - x_o\|,$$

Semantic Loss: The Visual Loss associates the Camouflager's output with the hijackee sample on the visual perspective. We now introduce the Semantic Loss which associates the Camouflager's output to the features of the hijacking sample. Since the Semantic Loss operates on the feature level and not the visual level, we first need a feature extractor $\mathcal{F}$ which extracts the features of a given sample. This feature extractor $\mathcal{F}$ can for example be a middle layer of any classification model. Since we do not assume the adversary's knowledge of any information about the target model as previously mentioned in Section II-D, we use a pretrained MobileNetV2 [29] as our feature extractor. However, a stronger adversary that has access to the target model can use the target model as the feature extractor. The output of any layer of $\mathcal{F}$ can be picked as the features. For our work, we use the output of the second to last layer of MobileNetV2.

Intuitively, the Semantic Loss calculates the L1 distance between the features of the output of the Camouflager and the hijacking sample. More formally, we define the Semantic Loss as follows:

$$\varphi_{sl} = \min\|\mathcal{F}(x_c) - \mathcal{F}(x_h)\|,$$

where $\mathcal{F}$ is the feature extractor.

Adverse Semantic Loss: So far the Visual and Semantic Losses already associate the Camouflager's output with both the visual appearance of the hijackee sample and the features of the hijacking sample. However, in certain cases when the hijacking and hijackee datasets are complex and similar, as

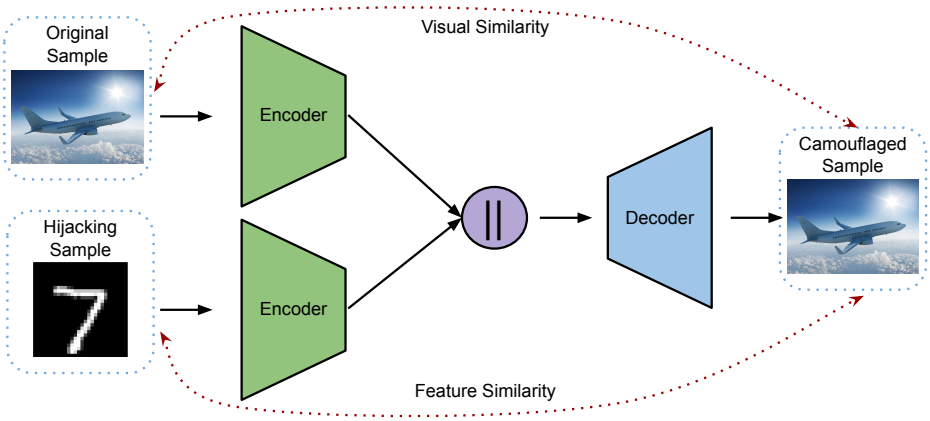


Fig. 1: A schematic view of the Camouflager. The encoders first encode the original and the hijacking samples. Then the outputs of the encoders are concatenated and inputted to the decoder. The decoder then generates the camouflaged sample which has the visual appearance of the original sample but the features of the hijacking one.

shown later, the camouflaged sample's features are not distinct enough from the hijackee sample's features, which degrades the performance of the model hijacking attack. Hence, we introduce another loss, i.e., the Adverse Semantic Loss. This loss maximizes the difference between the features of the hijackee and camouflaged samples using the L1 distance. We define the adverse Semantic Loss as:

$$\varphi_{asl} = \max \|\mathcal{F}(x_c) - \mathcal{F}(x_o)\|,$$

D. The Chameleon Attack

After presenting the general pipeline and the buildings blocks, we now present our first concrete model hijacking attack, namely the Chameleon attack.

Intuitively, the Chameleon attack uses a Camouflager to camouflage the hijacking dataset and poison the target model. The Chameleon attack can be divided into three stages, namely preparatory, camouflaging, and executing. We now explain each of them in detail.

Preparatory: In the first stage, the adversary setups their hijacking and hijackee datasets. To recap, this hijackee dataset is used for camouflaging the hijacking dataset.

Next, after creating the hijackee dataset, the adversary creates a mapping between the original dataset's labels and the ones in the hijacking dataset. In this work, we assign the labels in a non-semantic manner. More concretely, we assign the i^{th} label from the original dataset to the i^{th} label of the hijacking dataset, irrespective of what each label stands for. However, our attack is independent of the mapping technique and the adversary can freely create the mapping with the only restriction of keeping it consistent throughout the attack.

Finally, the adversary picks their feature extractor $\mathcal{F}$ which is used to calculate the features of samples needed to train the Camouflager. As previously mentioned (Section III-C), the feature extractor is an off-the-shelf model that the adversary can freely choose.

Camouflaging: After deciding on the hijackee dataset, label mapping, and the feature extractor, the adversary can now start the main process of the Chameleon attack. We use both of the Visual and Semantic Losses to build the Camouflager for this attack.

As previously mentioned and demonstrated in Figure 1, the Camouflager uses two encoders and a single decoder. All three models, i.e., the two encoders and the decoder, are trained jointly with both losses. More formally we define the loss as the following.

$$\mathcal{L}_{Cham}(x_c, x_o, x_h) = \min \Big(\|x_c - x_o\| + \|\mathcal{F}(x_c) - \mathcal{F}(x_h)\| \Big) \quad (1)$$

As the loss demonstrates, the Camouflager is independent of the target model. Hence, it can be used to hijack multiple target models with a similar original task.

For the Chameleon attack, the adversary uses the hijackee dataset and the hijacking dataset to train the Camouflager. More concretely, the adversary trains their Camouflager as follows:

1) For each epoch, the adversary first randomly pairs samples from the hijackee dataset to the samples in the hijacking dataset. Since both datasets can be of different sizes, the mapping between both datasets can be many-to-many instead of one-to-one. We change the mapping in each epoch to increase the generalizability of the Camouflager.

2) After mapping the samples, we feed each pair (a hijackee sample and a hijacking sample) to the Camouflager. Finally, the Camouflager's output and the input samples are used to update the Camouflager using Equation 1 as the loss function.

Executing: After training the Camouflager, the adversary can now execute their attack. Figure 2 shows an overview of the Chameleon attack after the training of Camouflager. As the figure shows, first, the adversary maps the samples inside the hijackee dataset to samples from the hijacking dataset and creates the camouflaged dataset by querying the trained Camouflager. To recap, the labels used to create the camouflaged dataset are the ones from the hijacking dataset. Next, they use

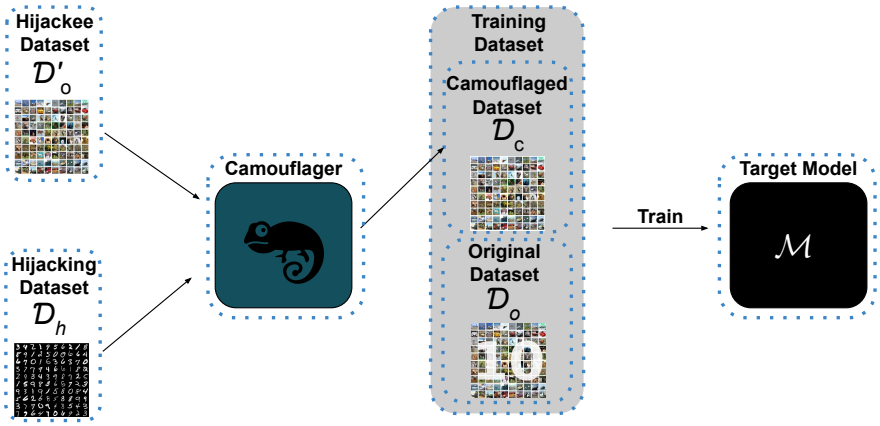


Fig. 2: An overview of the model hijacking attack. First, the adversary inputs the hijackee and hijacking datasets to the Camouflager. Next, they take the Camouflager's output (the camouflaged dataset) and poison the training dataset of the target model. Finally, the model is trained with the poisoned dataset.

the camouflaged dataset to poison the training of the target model to hijack it. We refer to the target model after being trained using the poisoned dataset as the *hijacked model*.

After hijacking the target model, the adversary can query any sample from the hijacking dataset's distribution by first camouflaging it using the Camouflager. Then, they query the camouflaged sample to the hijacked model, and map the predicted label to its corresponding label in the hijacking dataset.

E. The Adverse Chameleon Attack

As will be shown later in Section IV, the Chameleon attack has good performance when the hijacking and hijackee datasets are significantly different. However, when both datasets are complex and not significantly different, the performance starts degrading. Hence, we propose the more advanced attack, namely the Adverse Chameleon attack.

The Adverse Chameleon attack tries to explicitly distance the features of the output of the Camouflager from the hijackee dataset. To accomplish this, in addition to the Visual and Semantic Losses, we use the adverse Semantic Loss. More formally instead of using Equation 1 as the loss for training the Camouflager, we use the following loss.

$$\mathcal{L}_{ChamAdv}(x_c, x_o, x_h) = \min\Big(\|x_c - x_o\| + \|\mathcal{F}(x_c) - \mathcal{F}(x_h)\| - \|\mathcal{F}(x_c) - \mathcal{F}(x_o)\|\Big) \tag{2}$$

Besides the different loss function, to execute the Adverse Chameleon attack, the adversary follows the same steps as the Chameleon attack (Section III-E).

IV. Evaluation

In this section, we present our experimental results. We start by introducing our datasets and evaluation settings. Next, we evaluate our Chameleon and Adverse Chameleon attacks. Finally, we study the impact of some of the hyperparameters in our model hijacking attacks.

A. Datasets Description

To evaluate our different model hijacking attacks, we use three well-established computer vision benchmark datasets, namely MNIST, CIFAR-10, and CelebA. We now briefly introduce them:

MNIST: MNIST is a grey-scale handwritten digits classification dataset. It consists of 70,000 images, each of them is in the size of 28×28 and contains a single digit at its center. The MNIST dataset is equally split between 10 classes.

Since the state-of-the-art machine learning models we use in our work, e.g., the MobileNetV2 [29] and Resnet18 [11] models, expects inputs with the size of 224×224, we rescale the MNIST dataset to satisfy it. Moreover, we convert the grey-scale images to three channels images, by repeating the same values in all channels to be able to use the MNIST dataset on the same models trained on colored – three channels – images.

CIFAR-10: CIFAR-10 is a 10 classes colored dataset. It consists of 60,000 images with the size of 32×32. The images are equally split between the following 10 classes: Airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Similar to the MNIST dataset, we rescale the CIFAR-10 images to 224×224.

CelebA: CelebA is a dataset of face attributes with more than 200,000 colored images. We use the aligned version of the dataset, where each image contains the face of a celebrity in the middle of it and is labeled with 40 different binary attributes. We follow Salem et al. [27] to create an 8-class classification task by concatenating the top three balanced attributes, i.e., Heavy Makeup, Mouth Slightly Open, and Smiling. We randomly sample 40,000 images for training and 5,000 for testing. Finally,it is important to note that unlike the MNIST and CIFAR-10 datasets, the CelebA dataset is highly imbalanced.

B. Evaluation Settings

We now introduce our evaluation settings. We start with the model structures we use, then we present our evaluation metrics.

1) Model Structures: As previously mentioned, for this work, we focus on the machine learning classification setting. To this end, we use a state-of-the-art classification model for our target models (original task), namely Resnet18 [11].

Since we do not assume any knowledge about the target model for our model hijacking attacks, we use a completely different model as our feature extractor. More concretely, we use MobileNetV2 [29].

For the Camouflager, we use the following architecture for both of the encoders ($\mathcal{E}_o$ and $\mathcal{E}_h$):

The Camouflager encoders ($\mathcal{E}_o$ and $\mathcal{E}_h$) architecture:

$x_{in} \rightarrow$ `Conv2d(4,12)`
`Conv2d(4,24)`
`Conv2d(4,48)`
`Conv2d(4,96)`
$\rightarrow \mu$

Here, x_{in} is the input sample, `conv2d(k,f)` is a two dimensional convolution layer with kernel of size `k` and `f` filters, and μ is the encoder's output latent vector. After each convolution layer, we apply batch normalization and adopt `ReLU` as the activation function.

Finally, for the Camouflager's decoder we use the following architecture:

The Camouflager decoder ($\mathcal{E}^{-1}$) architecture:

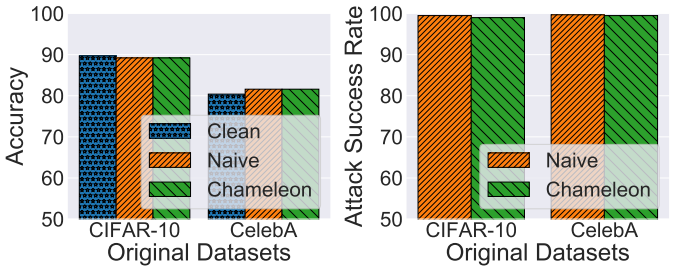
$(\mu_o||\mu_h) \rightarrow$
`ConvTranspose2d(4,96)`
`ConvTranspose2d(4,48)`
`ConvTranspose2d(4,24)`
`ConvTranspose2d(4,3)`
$\rightarrow x_{out}$

Here, $(\mu_o||\mu_h)$ is the concatenation of the latent vectors for both the original and hijacking samples after being encoded with the $\mathcal{E}_o$ and $\mathcal{E}_h$ encoders, respectively. `ConvTranspose2d(k',f')` is a two dimensional transposed convolution layer with kernel of size `k'` and `f'` filters, and x_{out} is the output camouflaged sample. After each layer, we apply batch normalization and adopt `ReLU` as the activation function, except for the last layer where we only use the `Tanh` activation function (to restrict the range for the decoded camouflaged sample).

Finally, we use the Adam optimizer to train the Camouflager.

2) Evaluation Metrics: To evaluate the performance of our model hijacking attack, we two metrics, namely *Utility* and *Attack Success Rate*.

Utility: Utility measures how close the performance of the hijacked model is to a clean, i.e., non-hijacked, model on the original dataset. The closer the performance of the hijacked and clean models, the better the model hijacking attack. More concretely, to measure the utility, we compare the accuracy of both the hijacked and clean models on a clean testing dataset, i.e., a testing dataset from the same distribution as the original dataset.



(a) Utility (b) Attack Success Rate

Fig. 3: The results of our Chameleon Attack. The original datasets are noted on the x-axis. For both (CIFAR-10 and CelebA) datasets, we use MNIST as the hijacking dataset. *Naive* corresponds to applying the model hijacking attack without camouflaging the hijacking dataset. Figure 3a compares the Utility of both the Naive and Chameleon attacks with a clean model (Clean) using the original testing dataset, and Figure 3b compares the Attack Success Rate of both attacks on the hijacking testing dataset.

Attack Success Rate: The Attack Success Rate measures the model hijacked attack performance on the hijacking dataset. We calculate the Attack Success Rate by computing the accuracy of the hijacked model on a hijacking testing dataset, i.e., a testing dataset with the same distribution as the hijacking dataset. For both of our Chameleon and Adverse Chameleon attacks, we first camouflage the hijacking testing dataset before querying it to the hijacked model and calculate the accuracy.

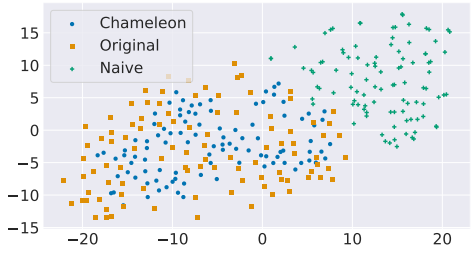
C. The Chameleon Attack

After introducing the datasets and our evaluations metrics, we now evaluate the performance of our Chameleon attack. We use MNIST as our hijacking dataset and both CIFAR-10 and CelebA as the original datasets for this attack.

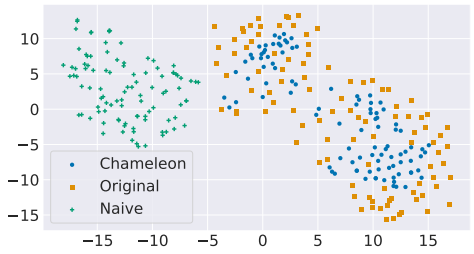
Firstly, we map the labels of the hijacking dataset and the original dataset as mentioned in Section III-D. Next, we train the Camouflager by constructing two encoders and a decoder with the architecture presented in Section IV-B1, and a hijackee dataset by randomly sampling $1,000$ sample for 8 random classes from the original dataset. Then we randomly sample $10,000$ samples from the hijacking dataset and follow the methodology previously presented in Section III-C to train the Camouflager.

After training the Camouflager, we use it together with the same hijackee dataset to camouflage $40,000$ randomly sampled samples from the hijacking dataset. Finally, we use the $40,000$ camouflaged samples together with their mapped labels to hijack the target model, i.e., we train the target model with both the camouflaged and original samples.

After presenting the concrete setup of our evaluation, we first evaluate the performance of the Chameleon attack (Requirement 1 and Requirement 3), then its stealthiness (Requirement 2 and Requirement 4).



(a) CIFAR-10



(b) CelebA

Fig. 4: Visualization of the difference in stealthiness between the Chameleon and Naive attacks. We use t-SNE to reduce the camouflaged, original, and hijacking samples to two dimensions. Then, we plot them in Figure 4a for the CIFAR-10 dataset, and Figure 4b for the CelebA dataset. Here MNIST is the hijacking dataset, and CIFAR-10 (Figure 4a) and CelebA (Figure 4b) are the original datasets.

Performance Evaluation: To evaluate the performance of our Chameleon attack, we consider the following baselines:

1) First, we train a clean model (Clean) using only the original dataset to compute and compare the Utility of the Chameleon hijacked models.

2) Second, we perform the naive model hijacking attack (Naive), where the adversary hijacks the target model without camouflaging the hijacking dataset first. It is important to note that this naive attack serves as the upper bound of the Attack Success Rate performance, since the hijacking samples are used as is without any modifications to make them less stealthy, which is the goal of our advanced model hijacking attacks.

We first compare the utility of our Chameleon attack in Figure 3a. As the figure shows, our Chameleon attack achieves almost the same Utility as both the Clean and Naive models. To recap, for reconstructing the Naive model we poison its training dataset with the hijacking datasets itself and not the camouflaged version. More concretely, our Chameleon attack achieves 89.2% accuracy on the original testing dataset, which is exactly the same as the Naive attack and only 0.5% lower than the Clean model for the CIFAR-10 dataset. For the CelebA dataset, our Chameleon and Naive hijacked models achieve 81.6% accuracy which is 1.2% better than the Clean model. We believe this improved performance is due to the regularization effect of the extra poison data. Similar improvement of the CelebA classification models after data poisoning has been previously observed in backdoor attacks [27].

Next, we compare the Attack Success Rate of our Chameleon attack. As the hijacking task here is MNIST, the Attack Success Rate measures the accuracy of the hijacking – MNIST – testing dataset. As Figure 3b shows, our Chameleon attack achieves almost the same performance as the Naive hijacked models. The Chameleon hijacked model achieves 99% Attack Success Rate when the original task is CIFAR-10 classification, which is only 0.5% lower than the Attack Success Rate of the Naive hijacked model. For the CelebA classification model, our attack achieves a 99.5% Attack Success Rate, which is only 0.2% lower than the one of the Naive model.

In general, hijacking models with the Chameleon attack can achieve almost perfect Attack Success Rate (Requirement 3) with a negligible drop in utility (Requirement 1), which shows the efficacy of this attack when the original and hijacking datasets are significantly different (as will be shown later in Figure 6).

Stealthiness Evaluation: Since two of the main requirements of our model hijacking attack focus on stealthiness (Requirement 2 and Requirement 4), we now compare the stealthiness of the Chameleon attack with the one of the Naive attack. We use the following two approaches to measure the stealthiness:

1) First, we measure the Euclidean distance between the hijacking and the original datasets, as well as the camouflaged and original datasets. To measure the Euclidean distance, we randomly sample 1,000 camouflaged, original, and hijacking samples. Then for each camouflaged/hijacking sample, we find the closest original sample to it, and calculate the Euclidean distance between them. We operate in a batch of 100 due to physical memory limitation. Finally, we average the Euclidean distances of the camouflaged and hijacking samples independently.

2) Second, we use the t-distributed stochastic neighbor embedding (t-SNE) [42] for reducing 100 samples from the hijacking, original, and camouflaged datasets to two dimensions. Then, we plot the reduced features of the samples.

First for the Euclidean distance, our experiments show that our Chameleon attack achieves 0.51 Euclidean distance when hijacking a CIFAR-10 classification model using the MNIST hijacking dataset, which is about 82% less than the one for the Naive attack (0.93). Similarly, for the CelebA classification model, our Chameleon attack achieves 0.77 Euclidean distance, which is about 56% less than the one of the Naive attack (1.2). A lower distance denotes a more stealthy attack, since it shows that the two datasets are more similar.

Second, we visualize the t-SNE reduced samples for the CIFAR-10 and CelebA hijacked models in Figure 4a and Figure 4b, respectively. As Figure 4 clearly shows, the camouflaged (Chameleon) samples are closer and hidden inside the original (Original) samples, unlike the hijacking (Naive) samples. Note that in the Naive model hijacking attack, the adversary poisons the training dataset of the target model using the hijacking dataset itself.

As both the Euclidean distance and the visualization in

(a) Original CIFAR-10 (b) Original CelebA (c) Original MNIST

(d) Camouflaged CIFAR-10 (e) Camouflaged CelebA

Fig. 5: Visualization of the output of the Camouflager for the Chameleon Attack for both the CIFAR-10 (Figure 5d) and CelebA (Figure 5e) datasets. Moreover, we show samples for both the Original(Figure 5a and Figure 5b) and hijacking (Figure 5c) datasets.

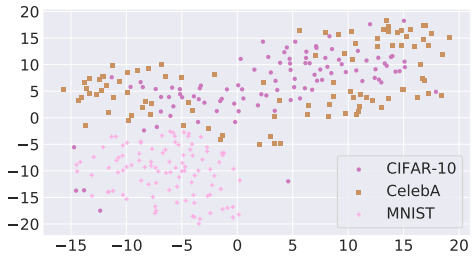
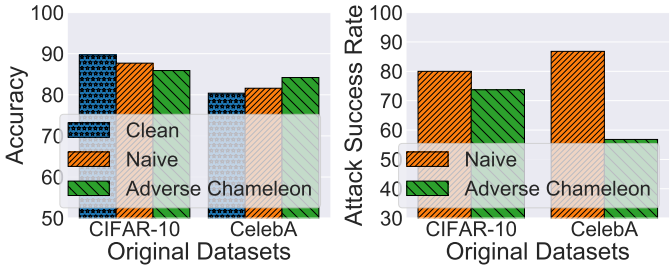


Fig. 6: Visualization of 100 random samples from the three datasets MNIST, CIFAR-10, and CelebA after reducing their dimension to two. As the figure shows, the CIFAR-10 and CelebA datasets are clustered together, while MNIST can be separated from them. This shows the hardness of using one of the CelebA or CIFAR-10 datasets to hijack the other, unlike using the MNIST dataset.



(a) Utility (b) Attack Success Rate

Fig. 7: The results of our Adverse Chameleon Attack. The original datasets are denoted on the x-axis, the hijacking dataset is CelebA when the original dataset is CIFAR-10 and vice versa. Naive corresponds to applying the model hijacking attack without camouflaging the hijacking dataset first. Figure 7a compares the utility of both the Naive and Adverse Chameleon attacks with a clean model using the original testing dataset, and Figure 7b compares the Attack Success Rate of both attacks on the hijacking testing dataset.

Figure 4 demonstrate, our Chameleon attack distinctly outperforms the Naive model hijacking attack in terms of stealthiness (Requirement 2).

Recall that our model hijacking attack has one more requirement with respect to the stealthiness of the attack (Requirement 4), i.e., predicting samples from the hijacking dataset randomly if they are not camouflaged. To this end, we run one more experiment to evaluate our Chameleon hijacked models using the hijacking testing dataset without camouflaging. Our results show that indeed the accuracy on the non-camouflaged testing dataset is around 10% for both cases, i.e., when using CIFAR-10 or CelebA as original datasets, which is the same as random guessing for the MNIST dataset

Finally, we visualize randomly sampled camouflaged samples together with ones from the original and hijacking datasets in Figure 5. Comparing figures Figure 5d and Figure 5e with figure Figure 5a and Figure 5b, we observe that indeed our camouflaged samples look like the original samples with some added artifacts. Moreover, it is clear that the camouflaged samples (Figure 5d and Figure 5e) are visually more similar than the hijacking samples (Figure 5c), when compared to the original samples (Figure 5a and Figure 5b).

D. The Adverse Chameleon Attack

As previously shown (Section IV-C), our Chameleon attack achieves strong performance when the hijacking and original datasets are significantly different. However, when performing the Chameleon attack using CIFAR-10 as the original dataset, and CelebA as the hijacking dataset, it only achieves an Attack Success Rate of 65.7% which is 14.3% less than the Naive attack. We believe this gap between the two attacks is due to the following two reasons:

1) The first one is related to the more complex nature of the CelebA dataset compared to MNIST. This can be seen in Figure 5, as the human faces have more information than the grey-scale digits.

2) Second, the CelebA and CIFAR-10 datasets are closer to each other compared to the MNIST dataset and either one of them. To visualize this, we randomly sample 100 samples from each dataset and reduce each of the samples using t-SNE to two dimensions.

Then, we plot the results in Figure 6. As the figure shows, the CelebA and CIFAR-10 datasets are clustered together and can be separated from the MNIST dataset.

Hence, when considering either of the CelebA or the CIFAR-10 datasets as the hijacking datasets, we execute the Adverse Chameleon attack. To evaluate the Adverse Chameleon attack, we follow the same evaluation settings previously introduced in Section IV-C with the following exception: Instead of using the Chameleon attack to train the Camouflager, we use the Adverse Chameleon attack to train it. To recap, the Adverse Chameleon attack uses the additional Adverse Semantic Loss to train the Camouflager together with both of the Visual and Semantic Losses.

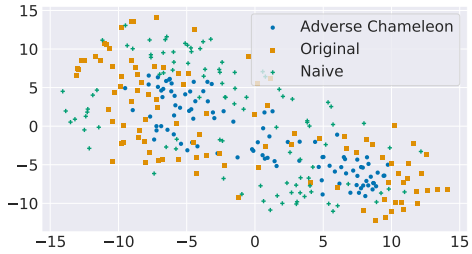
After introducing the concrete setup of the Adverse Chameleon attack we first evaluate its performance, then its stealthiness, and finally, we briefly discuss both the Chameleon and Adverse Chameleon attacks.

Performance Evaluation: We first compare the utility of our Adverse Chameleon attack in Figure 7a. As the figure shows, when using the Adverse Chameleon attack to camouflage the CelebA dataset and hijack a CIFAR-10 classification model, the utility is only slightly dropped. More concretely, the hijacked models achieve 87.7% and 85.9% accuracy on the CIFAR-10 testing dataset, when hijacking the models using the Naive and Adverse Chameleon attacks, respectively. This accuracy is only 2%, and 3.8% less than the one of a clean CIFAR-10 classification model. For the opposite case, i.e., the hijacking dataset is CIFAR-10 and the original dataset is CelebA, our Adverse Chameleon attack achieves 84.2% accuracy which is 2.6% and 3.8% higher than the one of the Naive attack and clean model, respectively. We believe this increase in performance is due to the regularization effect of our attack.

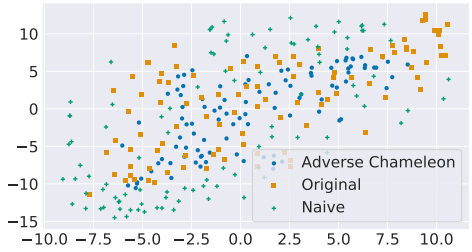
Next, we evaluate the Attack Success Rate of our Adverse Chameleon attack in Figure 7b. Specifically, we calculate the Attack Success Rate as the accuracy of the hijacking testing dataset when evaluating the Naive attack, and the camouflaged hijacking testing dataset when evaluating the Adverse Chameleon attack. As the figure shows, the Naive and Adverse Chameleon attacks achieve 80.0% and 73.7% Attack Success Rate when hijacking a CelebA classification model using the CIFAR-10 classification as the hijacking task, respectively. For the other case, when the adversary aims to hijack a CIFAR-10 classification model to perform CelebA classification task, our Adverse Chameleon attack achieves 56.8%, which is less than the one of Naive attack (86.8%) but still significantly higher than random guessing. Note that for this case, our Adverse Chameleon attack achieves 2.6% better utility than the Naive attack.

As both Figure 7a and Figure 7b show, our Adverse Chameleon attack satisfies both of our performance related requirements, i.e., Requirement 1 and Requirement 3.

Stealthiness Evaluation: Similarly to the Chameleon attack, we evaluate the Adverse Chameleon hijacked model against the non-camouflaged hijacking testing dataset. As expected, the Adverse Chameleon hijacked models achieve nearly random performance for the non-camouflaged hijacking testing dataset (Requirement 4). For instance, when using the CelebA dataset to hijack CIFAR-10, the non-camouflaged hijacking testing dataset achieves less than 20% accuracy.



(a) CIFAR-10



(b) CelebA

Fig. 8: Visualization of the difference in stealthiness between Adverse Chameleon and Naive attacks. We use t-SNE to reduce 100 camouflaged, original, and hijacking samples. Figure 8a shows the result when using CIFAR-10 as the hijacking dataset and CelebA the original dataset, and Figure 8b shows the opposite case.

(a) Camouflaged CelebA

(b) Camouflaged CIFAR-10

Fig. 9: Visualization of the output of the Camouflager for the Adverse Chameleon Attack. We show the results when using the CelebA as the hijacking dataset and CIFAR-10 as the original dataset in Figure 8a, and vice versa inFigure 8b.

Next, we follow the same steps previously introduced in Section IV-C to visualize and compare the stealthiness (Requirement 2) of the Adverse Chameleon using the Euclidean distance and t-SNE.

First, we compare the Euclidean distance. Using our Adverse Chameleon attack to hijack a CIFAR-10 classification model with a CelebA classification – hijacking – task results in 0.52 Euclidean distance. This is less than the Euclidean

distance of the Naive attack by a factor of 2.5. Similarly, when using the CelebA classification task as the original task and the CIFAR-10 classification as the hijacking task, our Adverse Chameleon achieves 0.77 Euclidean distance, which is 1.6 times lower than the one with the Naive attack.

Second, we visualize the t-SNE reduced samples for all the original, hijacking, and camouflaged samples in Figure 8. In Figure 8, the Naive samples directly correspond to the hijacking samples, since to perform the Naive attack, the adversary uses the hijacking samples themselves to poison the target model's training dataset. We show the result of using CelebA as the hijacking dataset and CIFAR-10 as the original dataset in Figure 8b, and vice verse in Figure 8a. As both figures show, the hijacked samples (Adverse Chameleon) are more clustered with the original samples (Original) than the non-camouflaged hijacking samples (Naive). Comparing the t-SNE results from the Chameleon attack (Figure 4) and the Adverse Chameleon attack (Figure 8) can further confirms that using the CelebA or CIFAR-10 classification tasks as the hijacking ones are indeed harder than using MNIST; as the original samples in Figure 4 are more distant from the rest, compared to the ones in Figure 8.

Finally, we visualize randomly sampled camouflaged samples for both cases of using the CelebA/CIFAR-10 dataset to hijack a CIFAR-10/CelebA classification task in Figure 9a/Figure 9b. As the figures show, our camouflaged samples look visually similar to the original samples with some added artifacts from the hijacking.

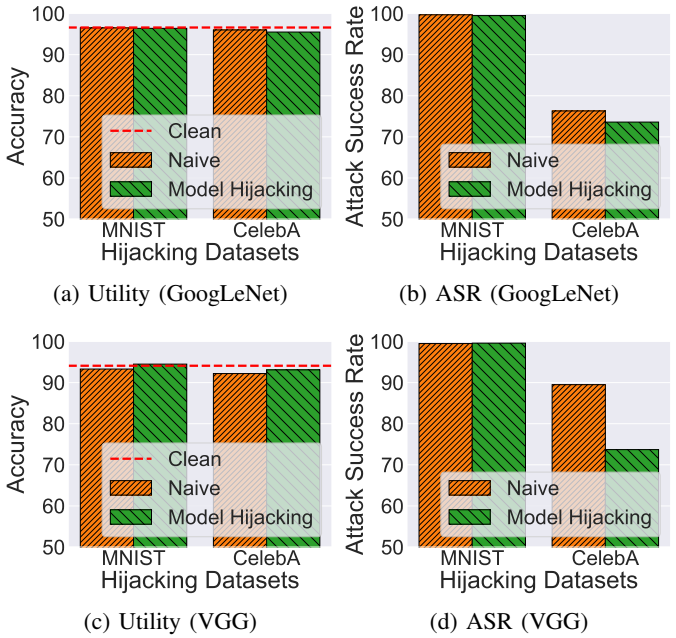
Discussion of Both Attacks: As demonstrated in this and the previous section (Section IV-C), both of our model hijacking attacks, i.e., the Chameleon and Adverse Chameleon attacks, achieve strong performance, i.e., they achieve a comparable Attack Success Rate when compared to the Naive attack, and similar utility compared to the clean models. Moreover, when the hijacking and original datasets are distinct, it is enough to use the Chameleon attack. However, when both datasets are more complex and similar, then the Adverse Chameleon attack is needed to enhance the performance the model hijacking attack.

E. Hyperparameters

We now explore some of the hyperparameters of our model hijacking attacks. We start by exploring the generalizability of our attack when using different target models and feature extractors. Next, we evaluate using different loss functions and the transferability of the Camouflager. Finally, we explore the effects of varying the hijackee dataset size and the poisoning rate on the model hijacking attack.

1) Different Target Models: We first evaluate the generalizability of our model hijacking attack on different target models. We use the CIFAR-10 dataset as our original dataset and evaluate the Chameleon and Adverse Chameleon attacks using MNIST and CelebA as the hijacking datasets, respectively.

We follow the previously mentioned setup in Section IV-C and Section IV-D to implement the Chameleon and Adverse Chameleon attack, with the exception of using GoogLeNet [36] and VGG16 [33] as the target models. To accelerate convergence, we use pretrained versions of the target models.



(a) Utility (GoogLeNet) (b) ASR (GoogLeNet)

(c) Utility (VGG) (d) ASR (VGG)

Fig. 10: The results of the Chameleon and Adverse Chameleon attacks when targeting a GoogLeNet (Figure 10a and Figure 10b) and VGG (Figure 10c and Figure 10d) based models. The hijacking datasets are denoted on the x-axis (MNIST for the Chameleon attack and CelebA for the Adverse Chameleon attack) and the original dataset is CIFAR-10. Moreover, we show the performance of the clean model using a red dashed line to compare the utility of both attacks.

Figure 10 shows the results for both target models. As the figure shows, both of our model hijacking attacks, i.e., Chameleon and Adverse Chameleon, achieves strong performance against the GoogLeNet and VGG target models. For instance, both attacks achieves a very similar accuracy similar to a clean model, i.e., the difference is less than 0.5% and 1% for the Chameleon and Adverse Chameleon, respectively; while achieving high ASR, i.e., above 99% for MNIST and 70% for CelebA.

Finally, compared to the Naive attack, our attack achieves similar performance, except for the CelebA case on the VGG target model. However, it is important to note that for this case, our attack achieves an improved performance with respect to utility.

These results show the generalizability of our model hijacking attack across different target models.

2) Different Feature Extractor: Second, we explore using different feature extractor to build the Camouflager and execute the model hijacking attack. To this end, we use the same evaluation setup similar to Section IV-E1 with the exception of using the same target models as Section IV-C and using the MnasNet [37] as the feature extractor. We select the MnasNet as it is significantly faster than the MobileNetV2 model. Finally, we present our results for using a pretrained MnasNet as our feature extractor in Table I.

As Table I shows, using the MnasNet achieves good perfor-

mance for both attacks (Chameleon and Adverse Chameleon). For instance, using the Chameleon attack, the hijacked model achieves 95.2% accuracy on the original dataset (Utility) and 80.7% accuracy on the hijacking one (ASR).

This shows the ability of the model hijacking attack to use different models as the feature extractor. As expected, using different models as the feature extractor can have different effects on the final performance of the model hijacking attack. However, we believe the model hijacking attack will have a strong attack performance as far as the feature extractor used has acceptable performance. We plan – in future work – to try using multiple feature extractors while training the Camouflager to further increase the independence of the model hijacking attack from the underlying feature extractor used.

TABLE I: The performance of the Chameleon and Adverse Chameleon attacks using MNIST and CelebA as hijacking datasets to attack a CIFAR-10 classification model, while using MnasNet as the Feature Extractor.

Hijacking Dataset	Utility	Attack Success Rate
MNIST	95.2	80.7
CelebA	92.8	60.5

3) Different Loss Functions: Next, we evaluate using different loss functions to implement our model hijacking attack. More concretely, we evaluate the effect of using the L2 instead of the L1 distance to implement our attack. We follow the same evaluation setup as Section IV-B1 with the exception of using L2 instead of L1 distance.

Our experiments show that using L2 distance achieves a strong performance as presented in Table II. For instance, using the CelebA dataset as the hijacking one to attack a CIFAR-10 classification model results in 86.1% accuracy and 63.2% ASR. This constitutes a drop in performance compared to when using the L1 loss with approximately 10% for the ASR, however, it improves the accuracy by 6.1%.

TABLE II: The performance of the Chameleon and Adverse Chameleon attacks using MNIST and CelebA as hijacking datasets to attack a CIFAR-10 classification model, while using L2 instead of L1 distance as the loss function.

Hijacking Dataset	Utility	Attack Success Rate
MNIST	90.1	99.5
CelebA	86.1	63.2

4) Transferability of the Camouflager: We now evaluate the transferability of the Camouflager. To this end, we use the previously trained Camouflagers used in Section IV-C and Section IV-D to hijack a CIFAR-100 classification model with MNIST and CelebA as the hijacking datasets, respectively. We use the pretrained Camouflagers to implement the Chameleon and Adverse Chameleon attacks as previously introduced in Section III-D and Section III-E, respectively.

Our experiments show that the Chameleon attack achieves 81.8% accuracy with a 99.5% ASR for the MNIST hijacking dataset. Similarly, the Adverse Chameleon attack achieves 78.6% accuracy with a 76.3% ASR for the CelebA hijacking dataset.

These results further demonstrate the transferability of the Camouflager after its training. In other words, the adversary can train a Camouflager and use it to hijack different models with different classification tasks.

5) Hijackee Dataset Size: We now evaluate the effect of the size of the hijackee dataset. Here, we use our Adverse Chameleon attack to hijack a CIFAR-10 classification model with the CelebA dataset.

We evaluate a range of different sizes for the hijackee dataset, namely we set the size to $10, 100, 1,000$, and $10,000$ samples. For each setting, we hijack the CIFAR-10 model and calculate both metrics,i.e., Utility and Attack Success rate.

Executing the Adverse Chameleon attack using hijackee dataset with the size of 10, 100, $1,000$ and $10,000$ achieves 44.7%, 60.5%, 73.7 and 65.8% Attack Success Rate, with an accuracy of 82.4%, 87.4%, 85.9% and 87.4%, respectively. As the results show, using a hijackee dataset of 10 samples is too small for executing the model hijacking attack. However, setting the size to 100 samples or more is already enough for the Adverse Chameleon hijacking attack. We select the hijackee with a size $1,000$ as it achieves the best overall performance compared to the other two.

6) Poisoning Rate: Next, we evaluate the effect of varying the poisoning rate on our model hijacking attack. In other words, we use different sizes of the hijacking dataset. To this end, we evaluate both of our Chameleon and Adverse Chameleon attacks while setting the size of the hijacking dataset (poisoning data) from $10,000$ to $40,000$ with a step of $10,000$. For both attacks, we set the original task to CIFAR-10 classification. For the hijacking task, we use MNIST classification for the Chameleon attack and CelebA classification for the Adverse Chameleon attack.

Our results show that for the simple case of using MNIST as the hijacking dataset, $10,000$, i.e., a poisoning rate of 17%, hijacking samples are enough for our Chameleon attack to hijack a CIFAR-10 classification model. More concretely, hijacking the model with $10,000$ hijacking samples results in the same Attack Success Rate (99%) as the hijacked model with $40,000$ samples, similarly, the difference between the utility of both models is negligible.

However, for the more complex task of using CelebA as the hijacking dataset to execute the Adverse Chameleon attack and hijack a CIFAR-10 classification model; the size of the hijacking dataset has a significant effect. For instance, the Attack Success rate is reduced from 73.7% to only 63.2% when using $20,000$ samples, i.e., a poisoning rate of 28%, instead of $40,000$. However, since there are fewer hijacking samples, the Utility of the hijacked model increases from 85.9% to 86.7%.

V. Related Works

In this section, we review some of the related works. We divide the related works into training and testing time attacks

against machine learning models. We start with the testing times attacks, then the training time ones.

A. Testing Time Attacks

Testing time attacks are the attacks executed by the adversary after the training of the model. We briefly review some of the related test time attacks against machine learning models.

Adversarial Reprogramming: One similar attack to our model hijacking attack is adversarial reprogramming [7]. Adversarial reprogramming is a test time attack, where the adversary optimizes a program to let the target model perform a different task. This program itself is an image with the target image padded inside to create the final input to the target model. The specially crafted input is then inputted to the target model, which performs the different task of classifying the padded image. The major difference between the adversarial reprogramming attack and our model hijacking attack is the different assumptions of the attacks, i.e., our model hijacking attack does not make any assumption about the target model and is a training time attack, but the adversarial reprogramming is a test time attack in which the adversary assumes knowledge of the target model similar to the assumptions needed for adversarial examples. Moreover, in the adversarial reprogramming attack, if the program is known, then the model can be easily patched since all images use the same program. However, for our model hijacking attack, the knowledge of any hijacked sample does not transfer to any other hijacked samples, i.e., there is no common feature/program for the hijacked samples.

Adversarial Examples: Adversarial examples [5, 15, 22, 23, 40, 43, 46] are a testing time attack where the adversary optimizes a noise such that when added to an image it gets misclassified. There exist two variants of the adversarial examples attack. The first is targeted adversarial examples, where the noise is optimized to classify the input sample to a specific label. The second is non-targeted ones, where the noise is optimized just to misclassify the input sample. Another field of work [13, 14, 20, 50] focus on using adversarial examples to enhancing the users/models privacy.

Other Attacks Against Machine Learning Models: There exist multiple other test time attacks such as: Membership inference [28, 32, 48] where the adversary aims at finding if a given sample was used into training the target model or not, Dataset reconstruction [26] where the adversary tries to reconstruct the updating dataset, Model stealing [19, 21, 41, 44] where the adversary tries to steal the target model, i.e., build a model with the same performance as the target model, and Model inversion [4, 8, 9, 51] where the adversary tries to reconstruct or complete some of the training samples of the target model.

B. Training Time Attacks

Training time attacks are the ones where the adversary executes their attack during or before the training of the target model. We briefly introduce some of the related training time attacks.

Data Poisoning Attack: Data poisoning attack [12, 34] is a training time attack where the adversary poisons the training dataset of the target model to compromise the model's utility. This poisoning of the training dataset is mostly done by flipping the ground truth of a subset of the dataset, such that the training of the target model fails. There are multiple works for poisoning different machine learning models/settings such as: Federated Learning [39], Support Vector Machines (SVM) [1], Regression Learning [12], Node Embeddings [2, 35], Next-Item Recommendation [49], and Neural Code Completion [30].

It is important to mention that our model hijacking attack can be adapted to any setting vulnerable to the data poisoning attack.

Backdoor Attack: The backdoor attack is another type of training time attacks, where the adversary manipulates the target model's training to backdoor it. The backdooring behavior is usually assigned with a trigger, which is when inserted in any input sample the target model predicts a specified label. Gu et al. [10] introduced BadNets the first backdoor attack against machine learning. BadNets uses a white square at the coroner of the images as a trigger to misclassify the backdoored inputs to a specific label. Salem et al. [27] later proposed dynamic backdoor, where instead of using a fixed trigger, they use a dynamic one. Another similar attack is the Trojan attack [16]. This attack simplifies the assumptions of the backdoor attack by not assuming the knowledge of any sample from the distribution of the target model's training dataset. There also exist multiple backdoor attacks attacking against Natural Language Processing (NLP) models [6], federated learning [45], video recognition [52], transfer Learning [47], and others [17, 24, 25, 38]

The backdoor attack can be considered a specific instance of the model hijacking attack by considering the classification of the backdoored samples as the hijacking dataset. However, our model hijacking attack is more general, i.e., it poisons the model to implement a completely different task.

VI. Discussion

In this section, we first discuss the limitations of our model hijacking attacks. Then, we review some of the possible defenses against them.

Limitation: The first limitation of our model hijacking attack is that the hijacking dataset cannot have more number of classes than the original one. To address this limitation, we propose to use a more complex hierarchical model hijacking attack with multiple virtual layers of classification tasks.

Intuitively, the adversary would start by grouping the hijacking dataset's classes into x clusters, where x is less than the number of classes of the original dataset. This constitutes the first layer of the hierarchical attack. Next, the adversary crafts a different backdoor-like trigger, i.e., a colored square at the corner of the input, for each cluster. To hijack a model, the adversary would need to poison its dataset with the following:

- Clean hijacking samples: Camouflaged samples without the triggers, with their corresponding cluster label, i.e., the first layer of the hierarchical attack.
- Triggered hijacking samples: Camouflaged samples with an added trigger on them. This trigger is specific

to which cluster this sample is from. The labels of these samples are set to their original ones modulo the target model's number of labels.

To execute the attack, the adversary uses the Camouflager to camouflage the sample, before querying it to the model. Then, depending on the output class, they add the corresponding trigger and query it again to the target model. We evaluate this hierarchical version using the Chameleon attack with MNIST and CIFAR-10 used as the hijacking and original datasets, respectively. Our experiments show that the utility of the hijacked model is not significantly affected. However, the ASR is significantly degraded to be lower than 30%. This shows the trade-off between having more labels than the original classification task and the attack performance. We plan to further explore different techniques – in future work – which would overcome this limitation with a better attack performance.

The second limitation of our attack is the visual – unnatural – artifacts on the camouflaged images. To address this limitation we propose different approaches. The first approach is to use a more powerful state-of-the-art autoencoder with more layers. However, that will come with the expense of increasing the cost of training the Camouflager. A different cheaper approach can be to combine multiple norms, e.g., the L^2 norm, when calculating the different losses. Moreover, we propose to use a weighting parameter to give more weight to the Visual Loss, hence making the output images more natural. Finally, a third approach is to add a discriminative model which penalizes the unnatural look of images. We plan to explore and evaluate these approaches in future work.

Finally, the third limitation of our attack is the cost of training the Camouflager. We recap that the Camouflager is only trained once at the start of the model hijacking attack, then is used during the training and after the deployment of the hijacked model. Moreover, once the Camouflager is trained, it can be used to hijack multiple target models performing a similar task as shown in Section IV-E4. However, since the training of the Camouflager can be computationally heavy. We propose to use a pretrained autoencoder and fine-tune it, which can reduce the training time. Moreover, we plan to explore – in future work – adapting few-shot learning techniques to further reduce the training cost of the Camouflager.

Possible Defenses: We now discuss some of the possible defenses against the model hijacking attack. A naive defense is adding noise to the images before inputting them to the model. This defense can degrade the attack performance, however, it will also degrade the performance of the original task. A more complex defense is using an autoencoder or different denoising techniques on the training and testing images. To evaluate this defense, we train an autoencoder on clean CIFAR-10 data and use it as a denoising step before querying the inputs to the target models hijacked with both the Chameleon and Adverse Chameleon attacks. Our results show that indeed using this step can reduce the ASR to almost random guessing, i.e., 11.1% for MNIST (Chameleon) and 18.4% for CelebA (Adverse Chameleon). However, it also significantly reduces the utility of the models. More concretely, the accuracy drops by 41.6% and 37.2% for the CelebA and MNIST datasets, respectively. We plan to further explore different defense techniques which can provide a better "defense utility" trade-off.

Another possible defense is to filter the outputs of the target model based on their entropy. In other words, the model owner first determines a threshold, and then calculate the entropy of each queried sample. If the entropy of this sample is above/below the threshold, then the model owner can accept/reject it. To evaluate this defense, we plot the distribution of the entropy for both clean and camouflaged samples and report the results in the Appendix (Figure 11). The distributions of both clean and camouflaged samples overlap, which would result in a high false-positive rate. Another challenge with this approach is determining an appropriate threshold. As the model owner needs to have access to both clean and camouflaged samples, which is a strong assumption in practice.

Generalization to Other Domains: As previously mentioned, we focus our model hijacking attack on computer-vision based machine learning models. However, we believe our attack can be extended to other domains. The most important requirement for the model hijacking attack is the ability to build a Camouflager. Intuitively, this means the ability to build an encoder-decoder model to transform the hijacking inputs to ones with similar features as the hijackee inputs.

VII. Conclusion

The continuous evolution of machine learning models has fueled the demand of including other parties in the training of the models, to be able to train the more complex emerging state-of-the-art models. An example of such machine learning paradigms is federated learning. This inclusion of new parties has opened new opportunities for adversaries to attack machine learning models. More concretely, an adversary can now participate in the training of a target model and manipulate the training process to implement their attack. This paradigm of machine learning attacks is referred to as the training time attacks.

In this work, we propose a new training time attack against computer vision based machine learning models namely, the model hijacking attack. In this attack, the adversary poisons the training dataset of a target model to hijack it into performing a hijacking task. This new type of attacks can cause severe security and accountability risks. Since the adversary can now hijack a benign model to perform an illegal or unethical task. Moreover, the hijacked model's owner can now be framed for the illegal or unethical task their model is capable of. Another risk of the model hijacking attack is parasitic computing, where the adversary can hijack a public accessible model to implement their private task, for saving the costs of training and maintaining their own model.

We propose two different model hijacking attacks, namely the Chameleon attack and the Adverse Chameleon attack. The Chameleon attack utilizes the Semantic and Visual Losses to hijack the target model, while the Adverse Chameleon attack in addition to these two losses, utilizes the Adverse Semantic Loss.

Our results show that indeed both of our model hijacking attacks (the Chameleon and Adverse Chameleon attacks) can efficiently hijack machine learning models. For instance, the

Chameleon attack achieves 99% Attack Success Rate on the hijacking task (MNIST classification) with only a utility drop of 0.5% on the original task (CIFAR-10 classification). Similarly, the Adverse Chameleon attack achieves 73.7% Attack Success Rate when hijacking a CIFAR-10 model with a CelebA classification – hijacking – task, with a utility drop of only 3.8%.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ ERC grant agreement no. 610150-imPACT, and from the Helmholtz Association within the project "Trustworthy Federated Data Analytics" (TFDA) (funding number ZT-I-OO1 4).

REFERENCES

[1] B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks against Support Vector Machines," in *International Conference on Machine Learning (ICML)*. icml.cc / Omnipress, 2012.

[2] A. Bojchevski and S. Günnemann, "Adversarial Attacks on Node Embeddings via Graph Poisoning," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 695–704.

[3] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," *CoRR abs/1902.01046*, 2019.

[4] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2019, pp. 267–284.

[5] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2017, pp. 39–57.

[6] X. Chen, A. Salem, M. Backes, S. Ma, Q. Shen, Z. Wu, and Y. Zhang, "BadNL: Backdoor Attacks Against NLP Models with Semantic-preserving Improvements," in *Annual Computer Security Applications Conference (ACSAC)*. ACSAC, 2021.

[7] G. F. Elsayed, I. J. Goodfellow, and J. Sohl-Dickstein, "Adversarial Reprogramming of Neural Networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[8] M. Fredrikson, S. Jha, and T. Ristenpart, "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 1322–1333.

[9] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2014, pp. 17–32.

[10] T. Gu, B. Dolan-Gavitt, and S. Grag, "Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," *CoRR abs/1708.06733*, 2017.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778.

[12] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2018, pp. 19–35.

[13] J. Jia and N. Z. Gong, "AttriGuard: A Practical Defense Against Attribute Inference Attacks via Adversarial Machine Learning," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2018, pp. 513–529.

[14] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 259–274.

[15] B. Li and Y. Vorobeychik, "Scalable Optimization of Randomized Operational Decisions in Adversarial Classification Settings," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR, 2015, pp. 599–607.

[16] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning Attack on Neural Networks," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[17] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks," in *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 182–199.

[18] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep Learning Face Attributes in the Wild," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 3730–3738.

[19] S. J. Oh, M. Augustin, B. Schiele, and M. Fritz, "Towards Reverse-Engineering Black-Box Neural Networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[20] S. J. Oh, M. Fritz, and B. Schiele, "Adversarial Image Perturbation for Privacy Protection – A Game Theory Perspective," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 1482–1491.

[21] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff Nets: Stealing Functionality of Black-Box Models," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 4954–4963.

[22] N. Papernot, P. D. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical Black-Box Attacks Against Machine Learning," in *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, 2017, pp. 506–519.

[23] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *IEEE European Symposium on Security and Privacy (Euro S&P)*. IEEE, 2016, pp. 372–387.

[24] A. S. Rakin, Z. He, and D. Fan, "TBT: Targeted Neural Network Attack with Bit Trojan," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 13 198–13 207.

[25] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden

Trigger Backdoor Attacks," in *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2020, pp. 11 957–11 965.

[26] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2020, pp. 1291–1308.

[27] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang, "Dynamic Backdoor Attacks Against Machine Learning Models," *CoRR abs/2003.03675*, 2020.

[28] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.

[29] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 4510–4520.

[30] R. Schuster, C. Song, E. Tromer, and V. Shmatikov, "You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion," *CoRR abs/2007.02220*, 2020.

[31] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2018, pp. 6103–6113.

[32] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2017, pp. 3–18.

[33] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[34] O. Suciu, R. Mărginean, Y. Kaya, H. D. III, and T. Dumitraş, "When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2018, pp. 1299–1316.

[35] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song, "Data Poisoning Attack against Unsupervised Node Embedding Methods," *CoRR abs/1810.12881*, 2018.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 1–9.

[37] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 2820–2828.

[38] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu, "An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks," in *ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2020, pp. 218–228.

[39] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data Poisoning Attacks Against Federated Learning Systems," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2020, pp. 480–501.

[40] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," in *International Conference on Learning Representations (ICLR)*, 2017.

[41] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing Machine Learning Models via Prediction APIs," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2016, pp. 601–618.

[42] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, 2008.

[43] Y. Vorobeychik and B. Li, "Optimal Randomized Classification in Adversarial Settings," in *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*. IFAAMAS/ACM, 2014, pp. 485–492.

[44] B. Wang and N. Z. Gong, "Stealing Hyperparameters in Machine Learning," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2018, pp. 36–52.

[45] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. yong Sohn, K. Lee, and D. Papailiopoulos, "Attack of the Tails: Yes, You Really Can Backdoor Federated Learning," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2020.

[46] W. Xu, D. Evans, and Y. Qi, "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[47] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent Backdoor Attacks on Deep Neural Networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 2041–2055.

[48] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting," in *IEEE Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 268–282.

[49] H. Zhang, Y. Li, B. Ding, and J. Gao, "Practical Data Poisoning Attack against Next-Item Recommendation," *CoRR abs/2004.03728*, 2020.

[50] Y. Zhang, M. Humbert, T. Rahman, C.-T. Li, J. Pang, and M. Backes, "Tagvisor: A Privacy Advisor for Sharing Hashtags," in *The Web Conference (WWW)*. ACM, 2018, pp. 287–296.

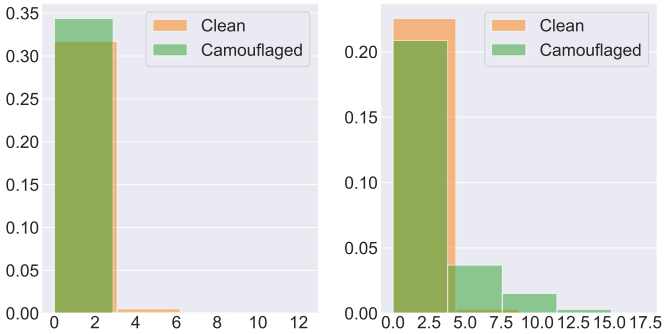
[51] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 250–258.

[52] S. Zhao, X. Ma, X. Zheng, J. Bailey, J. Chen, and Y.-G. Jiang, "Clean-Label Backdoor Attacks on Video Recognition Models," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 14 443–144 528.

APPENDIX

TABLE III: The description of the different datasets used in the model hijacking attack.

Dataset	Description
Original/Target Dataset	The dataset intended to be used by the target model's owner to train their model (the target model).
Hijackee Dataset	The dataset used to camouflage the hijacking samples, i.e., transform the visual appearance of the hijacking samples to ideally look alike the original dataset or make them harder to detect in general.
Hijacking Dataset	The dataset intended to be used by the adversary to hijack the target model.
Camouflaged Dataset	The hijacking dataset after being camouflaged by the Camouflager.
Poisoned Dataset	The dataset the model will train with, i.e., the concatenation of the camouflaged and the original datasets.



(a) MNIST (b) CelebA

Fig. 11: The distribution of the entropy of the outputs of the target model on the clean (CIFAR-10) and camouflaged (MNIST and CelebA) datasets. Figure 11a and Figure 11b show the results when hijacking a CIFAR-10 model using MNIST (Chameleon attack) and CelebA (Adverse Chameleon), respectively.