Saarland University

Department of Computer Science

Adversarial Inference and Manipulation of Machine Learning Models

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von
Ahmed Salem

Saarbrücken, 2022

Tag des Kolloquiums:	04 Oktober 2022
Dekan:	Prof. Dr. Jürgen Steimle

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Thorsten Herfet
Berichterstattende:	Prof. Dr. Michael Backes
	Dr. Yang Zhang
	Prof. Dr. Emiliano De Cristofaro
Akademischer Mitarbeiter:	Dr. Zhengyu Zhao

Zusammenfassung

Maschinelles Lernen (ML) hat sich als Kernkomponente für verschiedene kritische Anwendungen etabliert. Mit der zunehmenden Verbreitung von ML-Modellen sind jedoch auch zahlreiche Angriffe auf verschiedene Phasen der ML-Pipeline aufgetreten. Abstrakt betrachtet ist die ML-Pipeline in drei Phasen unterteilt, darunter Training, Update und Inferenz.

In dieser Arbeit werden die Datenschutz-, Sicherheits- und Verantwortlichkeitsrisiken der drei Phasen der ML-Pipeline bewertet. Zunächst untersuchen wir die Inferenzphase. Insbesondere untersuchen wir einen der schwerwiegendsten Angriffe auf ML-Modelle, nämlich den Membership Inference Attack (MIA). Wir lockern alle Hauptannahmen des MIA und zeigen, dass solche Angriffe mit geringen Kosten breit anwendbar sind und somit ein größeres Risiko darstellen als bisher angenommen. Zweitens untersuchen wir die Updatephase. Zu diesem Zweck führen wir eine neue Angriffsmethode gegen ML-Modelle ein, nämlich die Änderung der Ausgabe eines ML-Modells vor und nach dem Update. Anschließend stellen wir vier Angriffe vor, einschließlich auch Angriffe zur Datenrekonstruktion, die sich gegen dieses Szenario richten. Drittens untersuchen wir die Trainingsphase. In diesem Zusammenhang schlagen wir den Angriff “Model Hijacking” vor, bei dem der Angreifer das Zielmodell für seine eigenen illegalen Zwecke übernehmen kann. Schließlich schlagen wir verschiedene Verteidigungsmechanismen vor, um solche Risiken zu entschärfen.

Abstract

Machine learning (ML) has established itself as a core component for various critical applications. However, with this increasing adoption rate of ML models, multiple attacks have emerged targeting different stages of the ML pipeline. Abstractly, the ML pipeline is divided into three phases, including training, updating, and inference.

In this thesis, we evaluate the privacy, security, and accountability risks of the three stages of the ML pipeline. Firstly, we explore the inference phase, where the adversary can only access the target model after deployment. In this setting, we explore one of the most severe attacks against ML models, namely the membership inference attack (MIA). We relax all the MIA's key assumptions, thereby showing that such attacks are broadly applicable at low cost and thereby pose a more severe risk than previously thought. Secondly, we study the updating phase. To that end, we propose a new attack surface against ML models, i.e., the change in the output of an ML model before and after being updated. We then introduce four attacks, including data reconstruction ones, against this setting. Thirdly, we explore the training phase, where the adversary interferes with the target model's training. In this setting, we propose the model hijacking attack, in which the adversary can hijack the target model to provide their own illegal task. Finally, we propose different defense mechanisms to mitigate such identified risks.

Background of this Dissertation

This dissertation is based on the papers mentioned in the following. I contributed to all papers as the main author.

The initial idea of relaxing the key assumptions of the membership inference attack [P1] originated from a joint discussion between Mathias Humbert and Yang Zhang. Ahmed Salem and Yang Zhang then jointly designed the three different attacks presented in paper. For the two presented defenses, Ahmed Salem designed them. Ahmed Salem was then responsible for the implementation and evaluation. Michael Backes and Mario Fritz provided valuable feedback to the different stages of the design and evaluation of both the attacks and defenses. All authors participated in the writing and reviewing of the paper.

The idea of using the online learning as a new attack surface [P2] was proposed by Ahmed Salem and later refined to the different attacks by Yang Zhang. Ahmed Salem and Yang Zhang jointly designed the different attack and defense methodologies with the support of Michael Backes, Apratim Bhattacharya and Mario Fritz. The implementation and evaluation were done by Ahmed Salem. All authors participated in the writing and reviewing of the paper.

The general idea of exploring the accountability risks of machine learning models and the proposal of the model hijacking attack [P3] was the contribution of Ahmed Salem. The design, implementation, and evaluation were carried out by Ahmed Salem. All authors participated in the writing and reviewing of the paper.

[P1] Salem, A., Zhang, Y., Humbert, M., Berrang, P., Fritz, M., and Backes, M. 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.

[P2] Salem, A., Bhattacharya, A., Backes, M., Fritz, M., and Zhang, Y. Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning. In: *USENIX Security Symposium (USENIX Security)*. USENIX, 2020, 1291–1308.

[P3] Salem, A., Backes, M., and Zhang, Y. Get a Model! Model Hijacking Attack Against Machine Learning Models. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2022.

Further Contributions of the Author

The author was also able to contribute to the following:

Published Papers:

[S1] Chen, X., Salem, A., Backes, M., Ma, S., Shen, Q., Wu, Z., and Zhang, Y. BadNL: Backdoor Attacks Against NLP Models with Semantic-preserving Improvements. In: *Annual Computer Security Applications Conference (ACSAC)*. ACSAC, 2021.

[S2] Hanzlik, L., Zhang, Y., Grosse, K., Salem, A., Augustin, M., Backes, M., and Fritz, M. MLCapsule: Guarded Offline Deployment of Machine Learning as a Service. In: *CVPR Workshop on Fair, Data Efficient and Trusted Computer (TCV)*. IEEE, 2021.

[S3] Jia, J., Salem, A., Backes, M., Zhang, Y., and Gong, N. Z. MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, 259–274.

[S4] Liu, Y., Wen, R., He, X., Salem, A., Zhang, Z., Backes, M., Cristofaro, E. D., Fritz, M., and Zhang, Y. ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models. In: *USENIX Security Symposium (USENIX Security)*. USENIX, 2022.

[S5] Salem, A., Berrang, P., Humbert, M., and Backes, M. Privacy-preserving similar patient queries for combined biomedical data. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)*. Sciendo, 2019, 47–67.

Technical Reports:

[T1] Salem, A., Backes, M., and Zhang, Y. Don't Trigger Me! A Triggerless Backdoor Attack Against Deep Neural Networks. *CoRR abs/2010.03282* (2020).

[T2] Salem, A., Sautter, Y., Backes, M., Humbert, M., and Zhang, Y. BAAAN: Backdoor Attacks Against Autoencoder and GAN-Based Machine Learning Models. *CoRR abs/2010.03007* (2020).

[T3] Salem, A., Wen, R., Backes, M., Ma, S., and Zhang, Y. Dynamic Backdoor Attacks Against Machine Learning Models. *CoRR abs/2003.03675* (2020).

[T4] Salem, A., Wen, R., Backes, M., and Zhang, Y. Property Inference Attacks Against Black-box NLP Models (2021).

Acknowledgments

First and foremost, I would like to thank my advisor Michael Backes for giving me the opportunity to pursue my Ph.D. studies at CISPA. I am very grateful for his support and guidance throughout my studies.

I would also like to express my gratitude towards my dissertation committee members: Michael Backes, Emiliano De Cristofaro, and Yang Zhang, for their effort and time reviewing this thesis.

I am also very much thankful to thank Yang Zhang for helping and guiding me from the start of my Ph.D. till its end. More generally, I am deeply grateful to all my collaborators and co-authors that I was lucky enough to work with. Thank you all for the very interesting and insightful discussions.

Naturally, I want to thank all of my colleagues and friends inside and outside CISPA who made this journey fun! Special thanks go to Abdullah, Alfu, Rui, Akram, Shehy, Amr, Gad, Gilani, Zayat, and many more!

I would also like to acknowledge Gilani, Kathrin, Patrick, Rui, and Shehy for helping me to review parts of this thesis.

Most importantly, I would like to thank my family. My parents and sisters for their unlimited love and support, no matter how crazy my decisions are. The same applies to my wife, Heidi, who was always there to support me at all times. Also, my daughter, Dalida, who manages to make me happy independent of the situation or time.

Finally, for anyone whom I forgot to mention here due to my memory, I am really grateful and thankful for each and every person who helped me through this long journey, thank you all!!!

Contents

List of Figures

List of Tables

1 Introduction

Machine learning (ML) has progressed rapidly during the past decade. This progress has led to its adoption in a wide range of real-world applications, including those that play critical roles. For instance, machine learning models are now used in malware and spam detection [25], autonomous driving [14], and helping courts to take parole decisions [1]. Moreover, most of the current smart devices that are used day-to-day by users have artificial intelligence (AI) based systems that use machine learning.

Two of the most significant demands fueled by this increasing rate of machine learning adoption are the need for high-quality training data and high computational power. Such high demand for data and computational power hinder individuals from training ML models on their own. This has lead leading Internet companies to deploy machine learning as a service (MLaaS), e.g., Google [1], Amazon[2], and Microsoft [3]. Under such services, a user uploads their own dataset to a server and the server returns a trained ML model to the user, typically as a black-box API. Alternatively, users can collaborate and use new training paradigms, e.g., federated learning [16], which involve multiple parties jointly building an ML model such as

Despite being popular, ML models are vulnerable to various security and privacy attacks. These attacks can be classified into two categories depending on when they are performed. The first category is testing time attacks, where the adversary performs their attacks after the training of the target model. Some examples for testing time attacks are model inversion [29], adversarial examples [35], membership inference attack [75] and model extraction [87]. The second category is training time attacks. In these attacks, the adversary manipulates the training of an ML model. Training time attacks are intuitively due to the inclusion of new parties in the training process of ML models. Some examples in this domain include backdoor [36] and data poisoning attacks [13].

1.1 Our Contributions

In this dissertation, we evaluate the privacy and security of the ML pipeline. Abstractly, the ML model's life cycle/pipeline is divided into three phases. The first phase is the *training phase*, which includes data collection and preprocessing, and model designing and training. Next, the second phase, namely the *inference phase*, is the one where the model is published or deployed and can be used by users. Finally, the third and last phase occurs when new data is acquired, and the model needs to be updated accordingly. In other words, the *updating phase* is the one where the model owner updates their model using an updating dataset.

More concretely, in this dissertation, we comprehensively investigate the security, privacy, and accountability risks of the three phases of the machine learning pipeline, i.e., the training, updating, and testing phases. We start by exploring the possible threats against each phase. Then, we propose different defense mechanisms to mitigate the discovered risks. Our work stretches across the following peer-reviewed publications [P1, P2, P3], each investigating a different machine learning pipeline stage.

[1]`https://cloud.google.com/ml-engine/`
[2]`https://aws.amazon.com/machine-learning/`
[3]`https://azure.microsoft.com/en-us/services/machine-learning-studio/`

Membership Inference Attack: We start in [P1] by exploring the inference phase of the machine learning pipeline. In this work, we investigate one of the most severe privacy attacks against machine learning models, namely the membership inference attack (MIA). The first membership inference attack [75] has shown that the extraction of information on the training dataset is possible in the MLaaS setting, which has severe security and privacy implications. However, the early demonstrations of the MIA feasibility have many assumptions on the adversary, such as using multiple so-called shadow models, knowledge of the target model structure, and having a dataset from the same distribution as the target model's training data. We relax all these key assumptions, thereby showing that the MIA is broadly applicable at a low cost, thus posing a more severe risk than previously thought. In this work, we present a comprehensive study on this emerging and developing threat using eight diverse datasets which show the viability of the proposed attacks across domains. In addition, we propose effective defense mechanisms against the membership inference attack that maintain a high level of utility of the ML models.

Dataset Inference and Reconstruction Attacks: In our second work [P2], we explore the online learning setting, where ML models are updated frequently with newly-collected data, i.e., the updating phase. In this setting, if an ML model is queried with the same set of data samples at two different points in time, i.e., before and after being updated, it will provide different results. In this work, we investigate whether the change in the output of a black-box ML model after being updated can leak information of the dataset used to perform the update, namely the updating dataset. More formally, our main research question for this work is: *Can different outputs of an ML model's two versions queried with the same set of data samples leak information of the corresponding updating dataset?* This constitutes a new attack surface against black-box ML models and such information leakage may compromise the intellectual property and data privacy of the ML model owner. We propose four attacks following an encoder-decoder formulation, which allows inferring diverse information of the updating dataset. Our new attacks are facilitated by state-of-the-art deep learning techniques. In particular, we propose a hybrid generative model (CBM-GAN) that is based on generative adversarial networks (GANs) but includes a reconstructive loss that allows reconstructing accurate samples. Our experiments show that the proposed attacks achieve strong performance. Finally, we also discuss possible mitigations against our proposed attacks.

Model Hijacking Attack: Our third and final work of this thesis [P3] investigates the security and accountability risks of the training phase. In this work, we propose a new training time attack against computer vision based machine learning models, namely the model hijacking attack. In this new attack, the adversary aims to hijack a target model for executing a different task than its original one without the model owner noticing. The model hijacking attack can cause accountability and security risks since a hijacked model's owner can be framed for having their 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 classify 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. Model hijacking attacks are launched in the same way as existing data poisoning attacks. However, one additional requirement of the model hijacking attack is stealthiness, 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 the Chameleon and Adverse Chameleon attacks, based on a novel encoder-decoder ML model, namely the Camouflager. Our evaluation shows that both of our model hijacking attacks achieve high attack success rate, with a negligible drop in model utility. To remedy this situation, we further propose different defense mechanisms.

1.2 Organization

The rest of this dissertation is organized as the following. We first present the needed preliminaries and background in Chapter 2. Next, we investigate the inference phase and propose our membership inference attack in Chapter 3. Chapter 4 presents our dataset inference and reconstruction attacks that exploit the updating phase. We then explore the security and privacy of the training phase and introduce our model hijacking attack in Chapter 5. Finally, Chapter 6 presents the related work, and Chapter 7 concludes the dissertation.

2 Preliminaries and Background

In this chapter, we present the preliminaries needed for this thesis. We start by introducing the machine learning classification and online learning settings. We then introduce different basic attacks against machine learning models, including membership inference and data poisoning attacks. Finally, we present the different datasets used for our evaluation.

2.1 Machine Learning Classification Setting

In this thesis, we concentrate on the machine learning classification setting, as it is the most common machine learning application. A machine learning classifier aims to classify a data sample to a certain label/class. More formally, an ML classifier is essentially a function $\mathcal{M}$ that maps a data point $\mathbf{x}$ (a multidimensional feature vector) to an output vector, i.e.,

$$\mathcal{M}(\mathbf{x}) = \mathbf{y}$$

The size of $\mathbf{y}$, i.e., $|\mathbf{y}|$, denotes the number of unique labels ℓ. Each entry $\mathbf{y}_i$ in $\mathbf{y}$ represents the confidence of the model $\mathcal{M}$ assigning the label $\ell_i \in \ell$ to the data point $\mathbf{x}$. The parameters of an ML model are learned on a training dataset (denoted by $\mathcal{D}_{Train}$) containing multiple data points following a predefined learning object and an optimizer such as Adam.

2.2 Online Learning Setting

The online learning settings enables a trained ML model ($\mathcal{M}$) to be updated with an updating dataset (denoted by $\mathcal{D}_{Update}$). The model update is performed by further training the model with the updating dataset using the same optimization algorithm on the basis of the current model's parameters. More formally, given an updating dataset $\mathcal{D}_{Update}$ and a trained ML model $\mathcal{M}$, the updating process $\mathcal{F}_{Update}$ can be defined as:

$$\mathcal{F}_{Update} : \mathcal{D}_{Update}, \mathcal{M} \rightarrow \mathcal{M}'$$

where $\mathcal{M}'$ is the updated version of $\mathcal{M}$.

2.3 Membership Inference Against Machine Learning Models

Membership inference attack in the ML setting emerges when an adversary aims to find out whether their target data point is used to train a certain ML model. More formally, given a target data point $\mathbf{x}_{Target}$, a trained machine learning model $\mathcal{M}$, and external knowledge of an adversary, denoted by $\mathcal{K}$, a membership inference attack can be defined as the following function:

$$\mathcal{A} : \mathbf{x}_{Target}, \mathcal{M}, \mathcal{K} \rightarrow \{0, 1\}$$

where 0 means $\mathbf{x}_{Target}$ is not a member of $\mathcal{M}$'s training dataset $\mathcal{D}_{Train}$ and 1 otherwise. The machine learning model $\mathcal{M}$ that the adversary targets is also referred to as the *target model.* As in the work of Shokri et al. [75], we assume the adversary only has black-box access to the target model, such as an MLaaS API. In other words, the adversary can only submit data points to the model $\mathcal{M}$ and obtain their probabilistic output, e.g., $\mathcal{M}(\mathbf{x}_{Target})$.

The membership inference attack is realized using a binary classifier ($\mathcal{A}$). Depending on the assumptions, the adversary can construct $\mathcal{A}$ in different ways, which will be presented in Chapter 3.

2.4 Data Poisoning Attack

Data poisoning attack [13, 80, 43, 73] is a training time attack against ML models. The typical goal of data poisoning attack is to jeopardize the accuracy/utility of the target model. 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 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 2.1.

2.5 Datasets Description

We now present the datasets used for the evaluation of our different attacks and defenses. We use multiple benchmark computer vision datasets such as MNIST, CIFAR-10, CIFAR-100, and CelebA. Furthermore, we use well-established text (News), location (Insta-NY, Insta-LA, and Location), and binary (Adult and Purchase) datasets.

MNIST: MNIST [2] is a grey-scale handwritten digit classification dataset. It consists of 70,000 images, each of them is 28×28 pixels and contains a single digit at its center. The MNIST dataset is equally split between 10 classes (the digits from 0 to 9).

CIFAR-10: CIFAR-10 [3] 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.

CIFAR-100: CIFAR-100 [3] is similar to the CIFAR-10. However, it is a 100 classes dataset. The images are equally split between the 100 classes, with each class having 600 images.

Insta-NY: Insta-NY [11] contains a sample of Instagram users' check-in data in New York. Each check-in represents a user visiting a certain location at a specific time. Each location is affiliated with a category; in total, there are eight different categories, including entertainment, food, bar, outdoor, professional, residence, store, and transportation. Our ML task for Insta-NY is to predict each location's category. We use the number of check-ins happened at each location per hour on a weekly basis as the location's feature vector. We further filter out locations with less than 50 check-ins, in total, we have 19,215 locations for the dataset.

Insta-LA: Insta-LA [11] is the same as Insta-NY, however, the check-in data are collected from Los Angeles.

CelebA: CelebA [51] 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 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 the CelebA dataset is highly imbalanced.

News: The News dataset (20 Newsgroups) [46] is one of the most common datasets used for text classification and clustering. The dataset consists of 20,000 newsgroup documents categorized into 20 classes. The number of data points belonging to each class is very similar, i.e., the dataset has a balanced class distribution. We preprocess the News dataset by first removing headers, footers, and quotes from the documents. We then build a TF-IDF matrix out of the raw documents and use it as our feature.

Face: The Face dataset (Labeled Faces in the Wild) [4] consists of about 13,000 images of human faces crawled from the web. It is collected from 1,680 participants with each participant having at least two distinct images in the dataset. In our evaluation, we only consider people with more than 40 images, which leaves us with 19 participants, i.e., 19 classes. The Face dataset is challenging for facial recognition, as the images are taken from the web and not under a controlled environment, e.g., a lab. It is also worth noting that this dataset is unbalanced.

Adult: The – Census Income – Adult dataset is based on the Adult Dataset from UC Irvine Machine Learning Repository [5]. It is used for predicting if a person makes over 50K per year or not. It consists of 48,842 samples, each of them contains 14 attributes. These features types vary i.e., boolean features like gender, to limited values features such as the native-country or even continuous ones like age.

Location: This dataset is constructed from the Foursquare Dataset [93]. The Foursquare Dataset consists of check-in data collected from Foursquare on a global scale. It contains millions of check-ins and venues; however, we follow [75] to first select data generated from the Bangkok area. Next, we remove users with less than 25 check-in, and venues with less than 100 visits. We then divide the Bangkok area into 0.5km x 0.5km blocks, and ignore any empty block. We consider a binary feature for each block indicating if the check-in was in this block or not, and a binary feature for each location type, i.e., Mall, Market, etc. The final dataset consists of 6,951 users with 466 binary features. Finally, we cluster the resulting rows into 30 classes. Our classification task is given a user which is described by 466 binary features, we try to predict in which cluster this user is in.

Purchase: This dataset is based on the Kaggle's "Acquire Valued Shoppers" Challenge [6]. The task of this challenge is to predict the repeat customers from all customers.

The dataset contains the transaction history of individuals, i.e., brand, purchase amount, etc. Following Shokri et al. [75], we adopt a clustering algorithm, namely K-means, to manually define classes. The numbers of classes include 2, 10, 20, 50, and 100, therefore, we extend the Purchase dataset into 5 datasets. For instance, Purchase-100 represents the Purchase dataset with 100 different classes. The classification task of this dataset is to determine the cluster of given inputs.

3 Membership Inference Attack

Inference Phase

3.1 Introduction

The inference phase of machine learning is one of the most practical settings for adversaries. Since in this phase, the adversary only needs access to the target model's output after finishing its training, which is the most adopted settings in practice, i.e., model owners usually only publish an API to use their service/model without disclosing the model itself. In this chapter, we study one of the most severe privacy attacks against this phase, namely the *membership inference attack*.

In the membership inference attack, an adversary aims to determine whether a data item (also referred to as a data sample) was used to train an ML model or not. Successful membership inference attacks can cause severe consequences. For instance, if a machine learning model is trained on the data collected from people with a certain disease, by knowing that a victim's data belong to the training data of the model, the attacker can immediately learn this victim's health status. Previously, membership inference has been successfully conducted in many other domains, such as biomedical data [10] and mobility data [66].

Shokri et al. [75] present the first membership inference attack against machine learning models. The general idea behind this attack is to use multiple machine learning models (one for each prediction class), referred to as *attack models*, to make membership inference over the *target model*'s output, i.e., posterior probabilities. Given that the target model is a black-box API, Shokri et al. propose to construct multiple *shadow models* to mimic the target model's behavior and derive the data necessary, i.e., the posteriors and the ground truth membership, to train the attack models.

There are two main assumptions made by Shokri et al. [75]. First, the attacker needs to establish multiple shadow models with each one sharing the same structure as the target model. This is achieved, for example, by using the same MLaaS that trains the target model to build the shadow models. Second, the dataset used to train shadow models comes from the same distribution as the target model's training data, this assumption holds for most of the attack's evaluation [75]. The authors further propose synthetic data generation to relax this assumption. However, this approach can only be applied to datasets containing binary features for efficiency reasons.

3.1.1 Contributions

These two assumptions are rather strong which largely reduce the scope of membership inference attacks against ML models. In this chapter, we gradually relax these assumptions in order to show that far more broadly applicable attack scenarios are possible. Our investigation shows that indeed, membership inference in ML can be performed in an easier way with fewer assumptions than previously considered. To remedy this situation, we further propose two effective defense mechanisms.

Membership Inference Attack. We study three different types of adversaries based on the design and training data of shadow models. As Table 3.1 illustrates, we hereby gradually relax the assumptions of the previous work until we arrive at model and data independent adversary.

Adversary 1. For the first adversary, we assume they have a dataset that comes from the same distribution as the target model's training data. Here, we concentrate on

Table 3.1: An overview of the different types of adversaries. ✓means the adversary needs the information while - indicates the information is not necessary.

Adversary type	Shadow model design		Target model's
	No. shadow models	Target model structure	training data distribution
Shokri et al. [75]	multiple	✓	✓
Our adversary 1	1	-	✓
Our adversary 2	1	-	-
Our adversary 3	-	-	-

relaxing the assumptions on the shadow models.

We start by using only one instead of multiple shadow models to mimic the target model's behavior. As shadow models are established through MLaaS, which implements the pay-per-query business model, using one shadow model notably reduces the cost of performing the membership inference attack.

Extensive experimental evaluation (we use a suite of eight different datasets ranging from image to text under multiple types of machine learning models) shows that with one shadow model and one attack model, the adversary can achieve a very similar performance as reported by Shokri et al. [75]. For instance, when the target model is a convolutional neural network (CNN) trained on the CIFAR-100 dataset [3], our simplified attack achieves 0.95 precision and 0.95 recall, while the attack with 10 shadow models and 100 attack models (as in the previous work [75]) has 0.95 precision and 0.94 recall.

Then, we relax the assumption that the shadow model is constructed in the same way as the target model, In particular, we show that training the shadow model with different architectures and parameters still yields comparable attack performance. Moreover, we propose a new approach for shadow model training, which frees the adversary from even knowing the type of ML models used by the target model.

Adversary 2. For this adversary, we assume they do not have data coming from the same distribution as the target model's training data. Also, the adversary does not know the structure of the target model. This is a more realistic attack scenario compared to the previous one.

We propose a *data transferring attack* for membership inference in this setting. Concretely, we train our single shadow model with a different dataset. This means the shadow model here is not used to mimic the target model's behavior but only to capture the membership status of data samples in a machine learning training dataset.

The main advantage of our data transferring attack is that the adversary does not need to query the target model for synthetic data generation. In contrast, the previous approach [75] requires 156 queries on average to generate a single data sample. This means our data transferring attack is far more efficient, less costly, and harder to be detected by the MLaaS provider.

Experimental results show that the membership inference attack still achieves a strong performance, with only a few percentage drop compared to the first adversary. More interestingly, we show that our data transferring attack even works between datasets belonging to totally different domains. For example, by training a shadow

model with the 20 Newsgroups text dataset [46], we are able to achieve 0.94 precision and 0.93 recall for attacking a target model trained on the CIFAR-100 image dataset.
Adversary 3. This adversary works without any shadow model, i.e., the attack only relies on the posteriors (outcomes) obtained from the target model when querying it with target data samples. No training procedure is required at all. We show that statistical measures, such as maximum and entropy, over the target model's posteriors can very well differentiate member and non-member data samples. To make a concrete membership inference, we propose a threshold-choosing method. Experiments show that such a simple attack can still achieve effective inference over multiple datasets.

All these experimental results show that membership inference can be performed in a much simpler and more efficient way, which further demonstrates the severe risks of ML models.
Defense. To mitigate the membership risks, we propose two defense mechanisms, i.e., *dropout* and *model stacking.*
Dropout. One reason behind membership inference attacks' effectiveness is the inherent overfitting nature of machine learning models. When an ML model faces a data sample that it was trained on, it returns a high posterior for one class compared to others. Therefore, to defend against membership inference, we use a classical approach adopted in deep learning, namely dropout, which aims at preventing overfitting. Dropout randomly deletes in each training iteration a fixed proportion of edges in a fully connected neural network model.

Experiments on multiple datasets show that dropout can be a very effective countermeasure against membership inference. On the CIFAR-100 dataset, dropout (with 0.5 dropout ratio) decreases the performance of our first adversary from 0.95 precision and 0.95 recall to 0.61 and 0.60, respectively. Moreover, it almost preserves the same utility as the initial target model: The target model's prediction accuracy only drops from 0.22 to 0.21 (CIFAR-100). As dropout serves as a regularizer, we observe that, for several learning problems, e.g., the Purchase-100 dataset [75], the target model's accuracy even improves after applying dropout. Therefore, these models improve in performance *and* resilience in membership inference attacks.
Model Stacking. Although the dropout mechanism is effective, it is specific to deep neural networks. For target models using other machine learning classifiers, we propose a second defense mechanism, namely model stacking. Model stacking is a major class of ensemble learning. In model stacking, multiple ML models are organized in a hierarchical way to prevent overfitting. In our case, we construct the target model with three different machine learning models. Two models are placed in the first layer directly taking the original training data as input, while the third model is trained with the posteriors of the first two models.

Through extensive experiments, we show that model stacking is able to significantly reduce the membership inference's performance. For instance, both precision and recall of the attack (adversary 1) drop by more than 30% on the CIFAR-100 dataset trained with model stacking. Meanwhile, the target model's prediction performance stays almost the same.

In summary, this chapter's contributions are as follows:

- We broaden the class of membership inference attacks by substantially relaxing the adversarial assumptions.

- We evaluate membership privacy threat under three different adversarial setups on eight diverse datasets, ultimately arriving at a model and data independent adversary. Extensive experiments demonstrate the severe membership privacy threat for machine learning models.
- We propose two defense mechanisms, namely dropout and model stacking, and demonstrate their effectiveness experimentally.

3.1.2 Organization

The rest of this chapter is organized as the following. Section 3.2, Section 3.3, and Section 3.4 present the threat models, attack methodologies, and evaluations of our three different types of adversaries, respectively. In Section 3.5, we introduce the two defense mechanisms. Finally, Section 3.6 concludes the chapter.

3.2 Towards Model Independent Membership Inference Attacks (Adversary 1)

In this section, we describe our first adversary considered for membership inference attack. For this adversary, we mainly relax the assumption on their shadow model design. In consequence, membership inference attack can be performed in a far more efficient and less costly way.

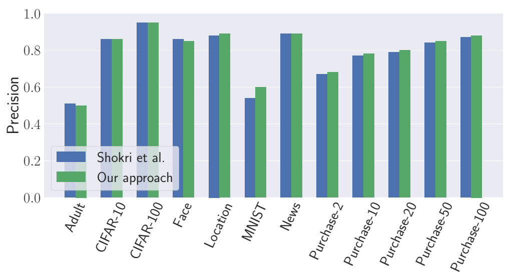
We start by defining the threat model. Then, we describe our first simplification, i.e., using one shadow model instead of multiple. In the end, we propose our second simplification which frees the adversary from knowing the target model's structure.

3.2.1 Threat Model

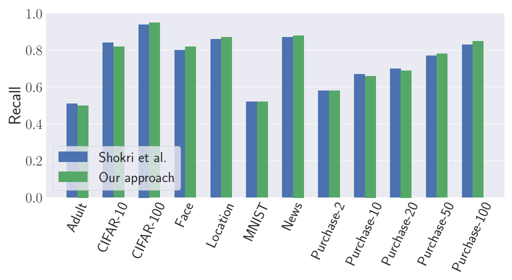
We define our attack model $\mathcal{A}$ as a supervised ML classifier with binary classes (member or non-member). To train $\mathcal{A}$, the adversary needs to derive the labeled training data. i.e., the ground truth membership. As mentioned in Section 2.3, the adversary only has black-box access to the target model, i.e., they are not able to extract the membership status from the target model. Therefore, the adversary trains a shadow model [75] to mimic the behavior of the target model, and relies on the shadow model to obtain the ground truth membership to train $\mathcal{A}$.

To train the shadow model, we assume that the adversary has a dataset, denoted by $\mathcal{D}_{Shadow}$, that comes from the same underlying distribution as the training data for the target model. Note that most of the experiments by Shokri et al. [75] make the same assumption.

We further assume that the shadow model uses the same ML algorithm and has the same hyperparameters as the target model. To achieve this in practice, the adversary can either rely on the same MLaaS provider which builds the target model or perform model extraction to approximate the target model [87, 60, 89]. Later in this section, we show this assumption can be relaxed as well.



(a) Precision.



(b) Recall.

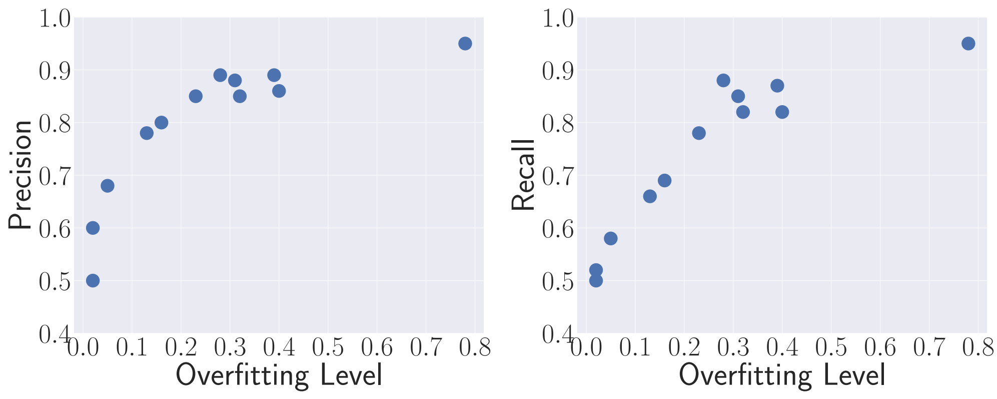
Figure 3.1: Comparison of the first adversary's performance with Shokri et al.'s using all datasets. (a) precision, (b) recall.

3.2.2 One Shadow Model

Methodology. The adversary's methodology can be organized into three stages, i.e., shadow model training, attack model training, and membership inference.
Shadow Model Training. The adversary first splits their dataset, i.e., $\mathcal{D}_{Shadow}$, into two disjoint sets, namely $\mathcal{D}_{Shadow}^{Train}$ and $\mathcal{D}_{Shadow}^{Out}$. Then, they use $\mathcal{D}_{Shadow}^{Train}$ to train their only shadow model, denoted by $\mathcal{S}$.
Attack Model Training. The adversary uses the trained shadow model $\mathcal{S}$ to perform prediction over all data samples in $\mathcal{D}_{Shadow}$ (consisting of $\mathcal{D}_{Shadow}^{Train}$ and $\mathcal{D}_{Shadow}^{Out}$), and obtain the corresponding posterior probabilities. For each data sample in $\mathcal{D}_{Shadow}$, they take its three largest posteriors (ordered from high to low) or two in the case of binary-class datasets as its *feature vector*. A feature vector is labeled as 1 (member), if its corresponding data sample is in $\mathcal{D}_{Shadow}^{Train}$, and as 0 (non-member) otherwise. All the generated feature vectors and labels are then used to train the attack model $\mathcal{A}$.
Membership Inference. To perform the attack on whether $\mathbf{x}_{Target}$ is in $\mathcal{D}_{Train}$, the adversary queries $\mathcal{M}$ with $\mathbf{x}_{Target}$ to obtain the corresponding posteriors. Then, they pick the 3 maximal posteriors, again ordered from high to low, and feed them into $\mathcal{A}$ to obtain the membership prediction.

It is important to note that our adversary only uses one shadow model and one attack model in their attack, while the approach by Shokri et al. [75] adopts multiple shadow models as well as multiple attack models (one for each class). In particular, as each shadow model is established through MLaaS [75], this strategy will largely reduce the cost of their membership inference attack.
Experimental Setup. We evaluate our attack over all datasets. For each dataset, we first split it by half into $\mathcal{D}_{Shadow}$ and $\mathcal{D}_{Target}$. Following the attack strategy, we split $\mathcal{D}_{Shadow}$ by half into $\mathcal{D}_{Shadow}^{Train}$ and $\mathcal{D}_{Shadow}^{Out}$. $\mathcal{D}_{Target}$, on the other hand, is used for attack evaluation, it is also split by half: One is used to train the target model, i.e., $\mathcal{D}_{Train}$, and serves as the members of the target model's training data, while the other serves as the non-member data samples.

For image datasets, i.e., MNIST, CIFAR-10, CIFAR-100, and Face, we use convolutional neural network (CNN) to build the target model. Our CNN is assembled with two convolutional layers and two pooling layers with one hidden layer containing 128 units in the end. For the other datasets, we use multilayer perceptron (neural network) with one



(a) (b)

Figure 3.2: The relation between the overfitting level of the target model measured by the difference between prediction accuracy on training dataset and testing dataset (x-axis) and membership inference attack performance (y-axis). (a) precision, (b) recall.

hidden layer (128 units) as the target model. Each shadow model's structure is the same as its corresponding target model, following the assumption that the adversary knows the target model's structure. The attack model is established with another multilayer perceptron (a 64-unit hidden layer and a softmax output layer). All our experiments are implemented in Python with Lasagne.[1] For reproducibility purposes, our code is available at `https://github.com/AhmedSalem2/ML-Leaks`.

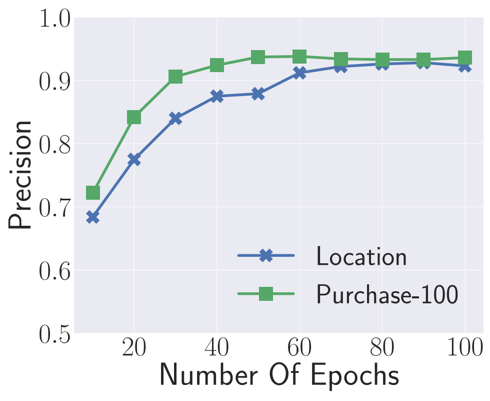
We compare our attack against the attack by Shokri et al. [75]. Following the original configuration of the authors' code[2], we train 10 shadow models and multiple attack models (one for each class).

As membership inference is a binary classification, we adopt precision and recall as our evaluation metrics. Moreover, we use accuracy to measure the target model's prediction performance.

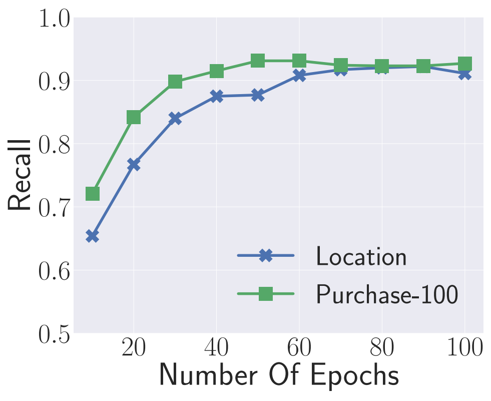
Results. Figure 3.1 depicts the first adversary's performance. In general, we observe that our attack has a very similar membership inference as the previous work [75]. For instance, our attack on the CIFAR-100 dataset achieves 0.95 for both precision and recall, while the attack by Shokri et al. has 0.95 precision and 0.94 recall. It is also interesting to see that our attack works for both balanced datasets, such as CIFAR-10, and unbalanced datasets, such as Face.

We also observe variations of the attack performance on different datasets. We relate this to the *overfitting level* of ML models on different datasets, similar to previous works [75, 95]. We quantify the overfitting level of a target model as the difference between its prediction accuracy on the training and testing datasets. Through investigation, we discover that if an ML model is more overfitted, then it is more vulnerable to membership inference attack (see Figure 3.2). For instance, our attack on the Adult dataset achieves a relatively weak performance (around 0.5 precision and recall), and there is only a 2% difference between the target model's training and testing accuracy.

[1]`https://github.com/Lasagne/Lasagne`
[2]`https://github.com/csong27/membership-inference`



(a)



(b)

Figure 3.3: The relation between the number of epochs used during the training of the target model (x-axis) and membership inference attack performance (y-axis). (a) precision, (b) recall.

On the other hand, the membership inference attack achieves 0.95 precision and recall on the CIFAR-100 dataset. Meanwhile, the corresponding target model provides a much better prediction performance on the training dataset than on the testing dataset, i.e., 78% difference.

To further demonstrate the relationship between overfitting and membership inference, we perform another – more controlled – experiment on the Location and Purchase-100 datasets. Concretely, we focus on the number of epochs used in training, larger number leads to a higher overfitting. We vary the number of epochs used from 10 to 100 and report the result in Figure 3.3. As we can see, the attack performance indeed increases with the increase of number of epochs.

Another factor which also affects our attack's performance is the number of classes in the dataset. Both CIFAR-10 and CIFAR-100 are similar image datasets with different number of classes (10 vs 100), it turns out that our membership inference attack on the latter dataset achieves a 10% better performance than on the former dataset. Similar results can be observed from all the Purchase datasets.

For our attacks, we use only the three highest posterior probabilities (in descending order) as the features for our attack. We test the effect of using more posteriors on the CIFAR-100, Location, MNIST, and News datasets. The result in Figure 3.4 shows that this factor does not have a significant effect on the attack's performance for most of the datasets. Generally, three posteriors achieves the best performance, especially on the MNIST dataset.

A major difference between our attack and the previous one [75] is the number of shadow models used. We further study this factor's influence on our own attack's performance. Figure 3.5 shows the corresponding results on the Purchase-100, Purchase-50, Adult, and Location datasets. By varying the number of shadow models from 1 to 10, we do not observe a significant performance difference for both precision and recall. This means increasing the number of shadow models does not improve our attack's performance.

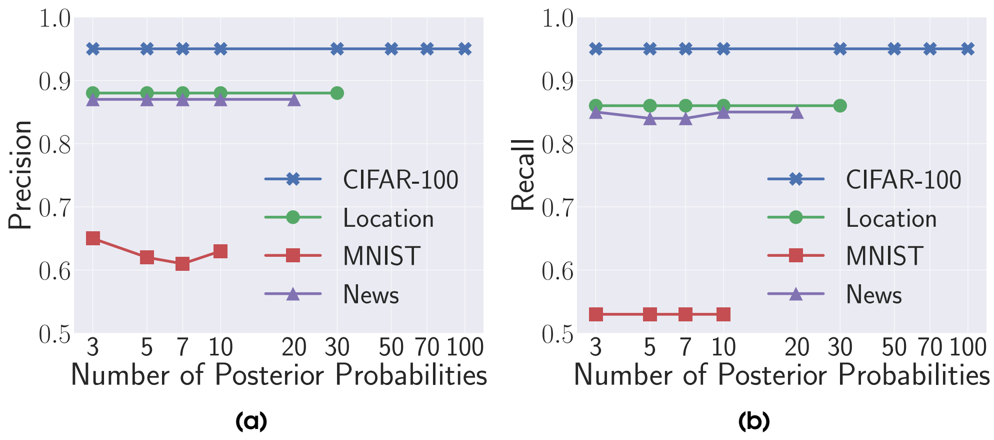


Figure 3.4: The effect of the number of posterior probabilities (used as features) on the first adversary's performance. (a) precision, (b) recall.

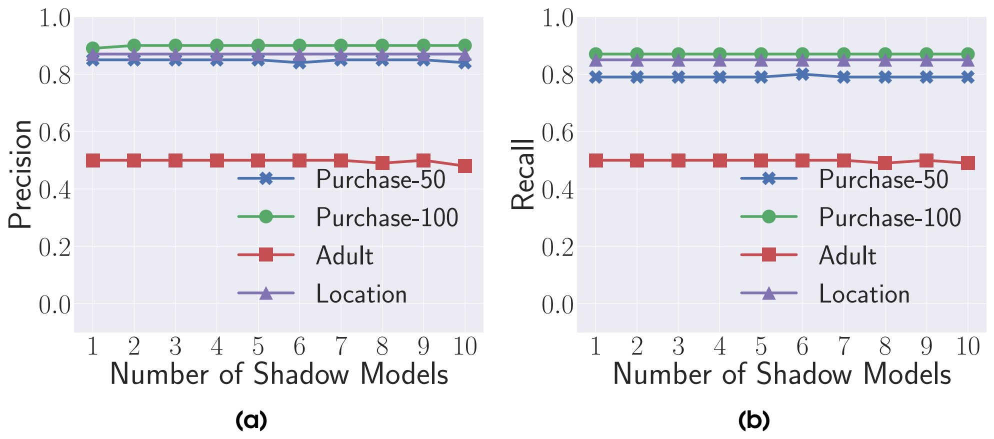


Figure 3.5: The effect of the number of shadow models on the first adversary's performance. (a) precision, (b) recall.

Evaluation on MLaaS. All the above experiments are conducted in a local setting. We further evaluate our attack with a real-world MLaaS. In particular, we use Google's MLaaS, namely Google Cloud Prediction API.[3] Under this service, a user can upload their own data and get the black-box ML API trained by Google. The user can neither choose which classifier to use, nor the corresponding model structure and parameters. We perform our attack following the same methodology as in Section 3.2.2. We construct both target model and shadow model with Google's MLaaS and build our attack model locally.

We use the Purchase-100 and Location datasets for evaluation and observe that the attack's performance is even stronger than our previous local evaluation. For the Purchase-100 dataset, our attack on Google's MLaaS has 0.90 precision and 0.89 recall,

[3]Google's Cloud prediction API is deprecated on April 20th, 2018 (`https://cloud.google.com/prediction/`), the new MLaaS provided by Google is called Cloud Machine Learning Engine.

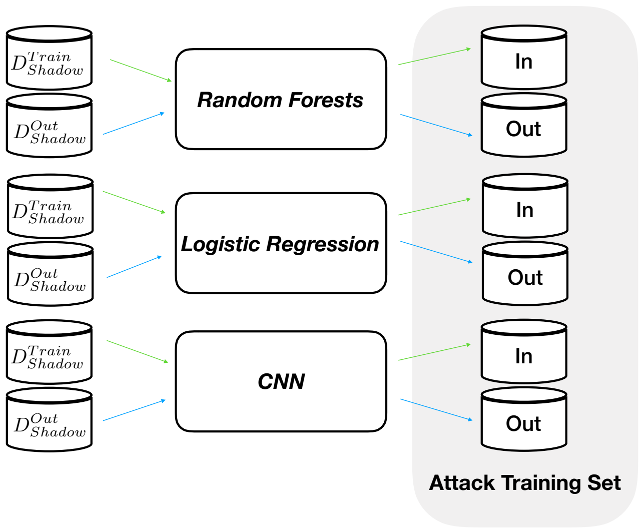


Figure 3.6: The architecture of the combining attack on generating data for training the attack model.

while our local evaluation has 0.89 precision and 0.86 recall. For the Location dataset, the precision is 0.89 and the recall is 0.86, which is almost similar to our local evaluation (0.88 precision and 0.86 recall).

3.2.3 Target Model Structure

One of the above attack's assumptions is that the adversary knows the target model's algorithm and hyperparameters and implements their shadow model in the same way. Next, we show how to relax this assumption. We first concentrate on target model's hyperparameters, then, the type of classifiers it uses.

Hyperparameters. We assume that the adversary knows the target model is a neural network, but does not know the details of the model. We first train the shadow model with half of the training parameters of the target model. More precisely, we reduce the batch size, hidden units, and regularization parameters to half. On the Purchase-100 dataset, our attack achieves 0.86 precision and 0.83 recall, which is almost the same as the one reported in Figure 3.1. We also revert the settings to test the case when the shadow model has double the number of parameters than the target model. The performance drops a bit to 0.82 precision and 0.80 recall, but it is still quite close to our original attack. We also perform evaluation over other datasets and observe similar results. This evaluation shows the flexibility of the membership inference attack: An adversary with no knowledge about the model's hyperparameters can still achieve good performance.

Target Model's Algorithm. We further assume that the adversary has no knowledge on what classification algorithm is adopted by the target model. In this setting, our first attempt is to use any classifier, such as random forests, as the shadow model and attack the target model that is (very likely to be) different from the shadow model, such

Table 3.2: Comparison of the combining attack and the original one, i.e., the first adversary proposed in Section 3.2.2.

Classifier	With target model structure		Combining attack	
	Precision	Recall	Precision	Recall
Multilayer perceptron	0.86	0.86	0.88	0.85
Logistic regression	0.90	0.88	0.90	0.88
Random forests	1.0	1.0	0.94	0.93

as CNN. However, the experimental results are not very promising.

To improve the attack with no knowledge of the target model, we construct a set of ML models, each with a different classification algorithm, and combine them together as one shadow model. Each single ML model is referred to as a *sub-shadow model.* This is achievable as the types of classifiers are limited. This attack, also referred to as the *combining attack*, can learn the behavior of the different classifiers and therefore can attack an unknown target model based on the assumption that there is a sub-shadow model which is trained with the same classifier as the target model.

Concretely, we use the same methodology as in Section 3.2.2 to train multiple sub-shadow models as illustrated in Figure 3.6, with each sub-shadow model being a different classifier. The data each sub-shadow model is trained on is the same. All the generated features by all sub-shadow models are stacked together, i.e., the attack model $\mathcal{A}$ is trained with a larger dataset. In this new dataset, each data sample in $\mathcal{D}_{Shadow}$ is represented multiple times with respect to the different sub-shadow models' outputs.

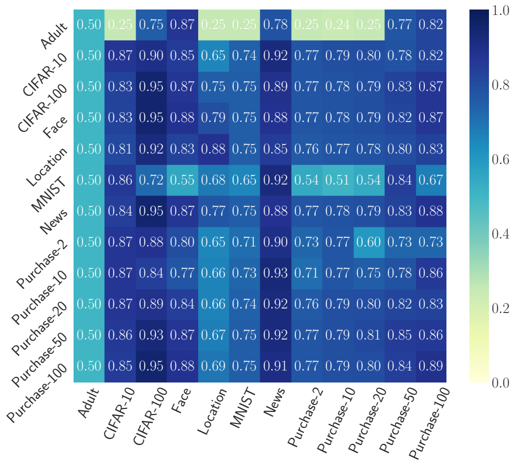
We run a local experiment on the Purchase-100 dataset to evaluate this attack. Three popular ML classifiers, i.e., multilayer perceptron, random forests (with 1,000 tree), and logistic regression are adopted as sub-shadow models. The target model for the Purchase-100 dataset is a multilayer perceptron. For a more complete comparison, we further build another two target models that are based on random forests and logistic regression, respectively, and use the same algorithm to build a single shadow model as in Section 3.2.2. Table 3.2 depicts the result. As we can see, our combining attack has a similar performance when target model is multilayer perceptron and logistic regression. Meanwhile, the attack's performance is relatively worse is when the target model is random forests.

In conclusion, we show that our combining attack can free the attacker from knowing the target model, which further enlarges the scope of membership inference attack.

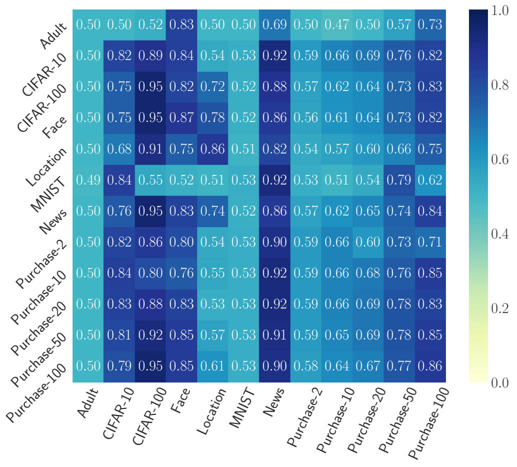
3.3 Towards Data Independent Membership Inference Attacks (Adversary 2)

In this section, we relax the assumption on the adversary having a dataset that comes from the same distribution as the target model's dataset.

We start by explaining the threat model, then describe the adversary's attack methodology. In the end, we present a comprehensive experimental evaluation.



(a) Precision



(b) Recall

Figure 3.7: The performance of our data transferring attack. The x-axis represents the dataset being attacked, i.e., the dataset the target model is trained on. The y-axis represents the dataset used for training the shadow model.

3.3.1 Threat Model

Different from the threat model in Section 3.2, we remove the assumption that the adversary has a dataset $\mathcal{D}_{Shadow}$ coming from the same distribution as the training dataset of the target model. This largely reduces the attack capabilities of the adversary. For this scenario, Shokri et al. [75] propose to query the target model multiple times to generate synthetic data for training the shadow model. However, this approach can only be applied when the dataset is assembled with binary features[4]. In contrast, our approach can be applied to attack ML models trained on any kind of data.

3.3.2 Methodology

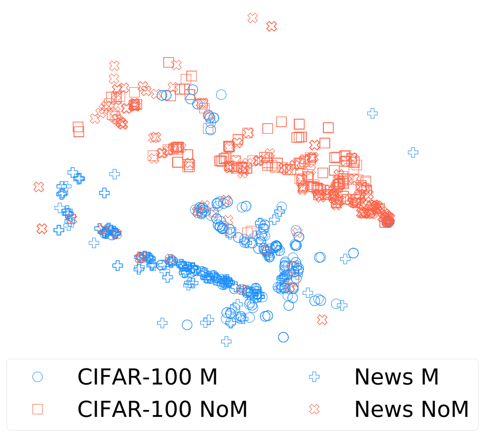
The strategy of the second adversary is very similar to the one of the first adversary. The only difference is that the second adversary utilizes an existing dataset that comes from a different distribution than the target model's training data to train their shadow model. We refer to this attack as the *data transferring attack*.

The shadow model here is not to mimic the target model's behavior, but to summarize the machine learning model's behaviour on member and non-member data. As only the three - or two in case of binary datasets - largest posteriors are used for the attack model, we can also neglect the effect brought by datasets with different number of classes.

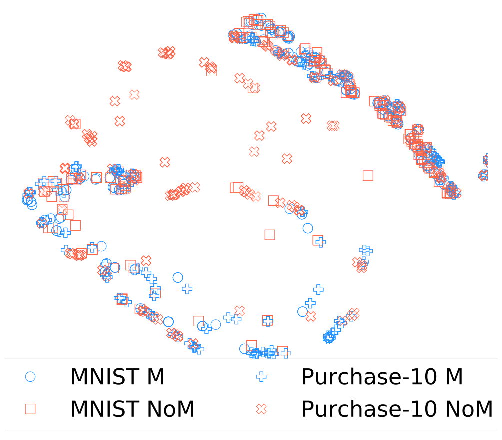
3.3.3 Evaluation

Experimental Setup. We use the same attack model and shadow model setup as presented in Section 3.2, such as data splitting strategy and the types of ML models

[4]We confirm this with the authors.



(a)



(b)

Figure 3.8: The top three posteriors of randomly sampled member and non-member data projected into a 2D space using t-Distributed Stochastic Neighbor Embedding (t-SNE). (a) CIFAR-100 and News, (b) MNIST and Purchase-10. M means member and NoM means non-member.

used. We perform the data transferring attack over all datasets. For evaluation metric, we again use precision and recall.

Results. Figure 3.7 depicts the data transferring attack's performance. The x-axis represents the dataset being attacked, i.e., the dataset the target model is trained on, and the y-axis represents the dataset used for training the shadow model. Compared to the first adversary, the attack results of which are listed at the diagonal of Figure 3.7, the second adversary in most cases obtains similar performances. For instance, using the Face dataset to attack the CIFAR-100 one results in 0.95 for both precision and recall, while the corresponding results for the first adversary are also 0.95 for both metrics. In several cases, we even observe a performance improvement over the first adversary. For instance, using the Purchase-10 dataset to attack the News dataset achieves 0.93 precision and 0.92 recall, while the first adversary has 0.88 precision and 0.86 recall. More interestingly, in many cases, datasets from different domains can effectively attack each other, e.g., the News and CIFAR-100 datasets.

For the first adversary, we relax the assumption on shadow model design. This relaxation also applies for the second adversary, as the shadow and target models are trained with different datasets. For instance, the Purchase-20 dataset is trained with a multilayer perceptron while the CIFAR-100 dataset is trained with a CNN.

One of the major advantages of our data transferring attack lies in its applicability. The synthetic data generation strategy by Shokri et al. [75] cannot be applied to dataset of any kind, but those with binary features. Even for dataset of binary features, a single synthetic data sample requires 156 queries to the target model [75]. Given the large dataset quantity needed for ML models and MLaaS's pay-per-query business model, this is very costly. Moreover, sending a large amount of queries to an MLaaS API would alert the server, which may not even allow the adversary to finish their synthetic data generation process. Meanwhile, our data transferring attack does not have any of the above constraints.

Reasoning. After demonstrating the strong performance of our data transferring attack, we now seek to understand the reason behind. To this end, we pick the highest three posteriors (similar to our attack) of member and non-member data samples with respect to their target ML models of all datasets, and embed these posteriors into a 2D space using t-Distributed Stochastic Neighbor Embedding (t-SNE). We show in Figure 3.8a the result for two datasets (of different types) between which our transferring attack is effective. As we can see, the member and non-member samples of these datasets are tightly clustered together and follow a common decision boundary, thus, the attack model trained on one dataset can effectively infer the membership status of samples in the other dataset. Meanwhile, Figure 3.8b shows the results for two datasets between which our transferring attack is not effective. As depicted, there are no clear clusters for members and non-member data samples.

3.3.4 Evaluation On MLaaS

We also evaluate our data transferring attack on Google's MLaaS. Concretely, we use a shadow model trained on the Location dataset to attack a target model trained on the Purchase-100 dataset. Both models are trained with Google's MLaaS. Experimental results show that we achieve 0.8 precision and 0.78 recall. By further flipping the shadow and target model, i.e., Purchase-100 dataset attacking Location dataset, the membership inference result is still very strong with 0.87 precision and 0.82 recall. This shows that our data transferring attack is not only effective in the local setting but also in the real-world MLaaS setting.

3.4 Model and Data Independent Membership Inference Attack without Training (Adversary 3)

In this section, we present our third adversary, who does not need to train any shadow model and does not assume any knowledge about the target model or data distribution. We start with the threat model description. Then, we list the attack methodology. In the end, we present the evaluation results.

3.4.1 Threat Model

We relax the assumption that the adversary needs to train a shadow model for performing their attack. All they could rely on are the target model's output posteriors $\mathcal{M}(\mathbf{x}_{Target})$ after querying their target data sample $\mathbf{x}_{Target}$. Note that Yeom et al. [95] propose a similar attack, however, their membership inference attack requires the adversary to know the target data sample's class label which is hard to obtain in some cases, such as in biomedical settings [12]. Therefore, our threat model covers a broader range of scenarios.

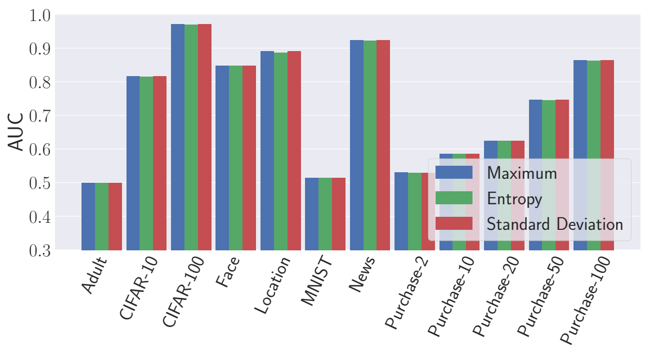


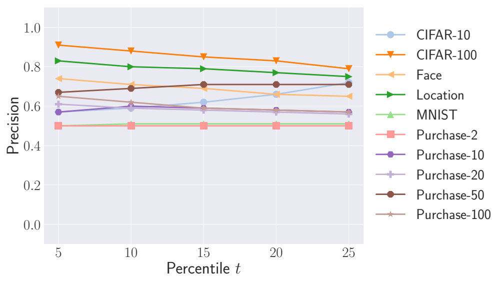
Figure 3.9: The AUC values for three different statistical measures over all datasets. We include the results for the Adult and News datasets as AUC is independent of a concrete detection threshold.

3.4.2 Methodology

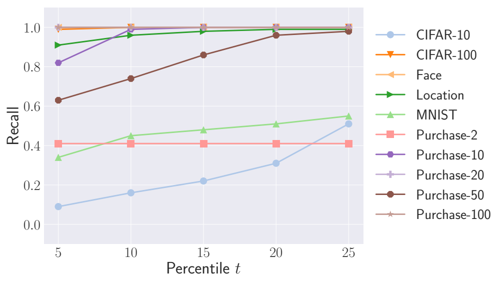
The attack model for the third adversary is implemented as an unsupervised binary classification. Concretely, the adversary first obtains $\mathcal{M}(\mathbf{x}_{Target})$. Then, they extract the highest posterior and compares whether this maximum is above a certain threshold. If the answer is yes, then they predict the data sample is in the training dataset of the target model and vice versa. The reason we pick maximum as the feature follows the reasoning that an ML model is more confident, i.e., one posterior is much higher than others, when facing a data sample that it was trained on. In another words, the maximal posterior of a member data sample is much higher than the one of a non-member data sample.

Threshold Choosing. The attacker can pick the threshold for membership inference depending on their requirements, as in many machine learning applications [99, 11]. For instance, if they concentrate more on inference precision (recall), then they can pick a relatively high (low) threshold.

Nevertheless, we provide a general method for choosing a threshold. Concretely, we generate a set of random samples in the feature space of the target dataset. For image datasets including CIFAR-10, CIFAR-100, MNIST, and Face, we generate random images, where the value of each pixel is drawn from a uniform distribution. For datasets with binary features including Location and Purchase datasets, we generate 0 and 1 for each feature according to an unbiased coin flip. For Adult and News, as the bounds for features are not clear, our method cannot apply. One way to tackle this is to collect News articles or people's records (with the same features as in the Adult dataset) from the Internet as the "random" samples. We leave this for future work. Next, we query these random samples to the target model to get the corresponding maximal posteriors. We hypothesize that these samples act as the non-member samples. Thus, top t percentile of these random samples' maximal posteriors can serve as a good threshold. Below, we show empirically that there exists a choice of t percentile that works well and generalizes across all the dataset and therefore can be used to automatically determine the detection threshold.



(a)



(b)

Figure 3.10: The relation between the percentile of the maximal posterior, i.e., threshold, (x-axis) and the third adversary's performance (y-axis). (a) precision, (b) recall.

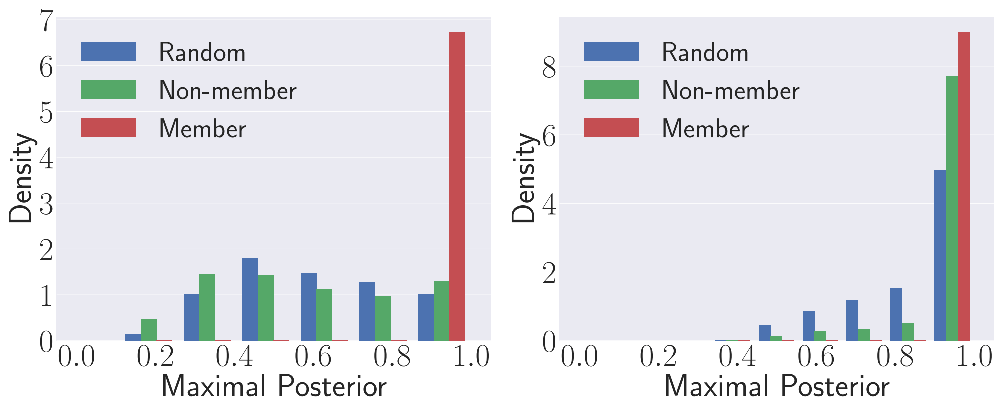
3.4.3 Evaluation

Experimental Setup. We evaluate the third adversary over all datasets except News and Adult. Note that we do not need to split the dataset as this adversary does not train any shadow model. Instead, we split each dataset by half, and use one part to train the target model and the other part is left out as non-members.

Results. We first evaluate the effectiveness of maximal posterior on differentiating member and non-member samples without setting a threshold. To this end, we adopt the AUC (area under the ROC curve) value, which reports the relation between true positive rate and false negative rate over multiple thresholds, as the evaluation metric [30, 11, 66, 63, 98]. Besides maximal posterior probability, we further test the effect of using other statistical metrics, including standard deviation and entropy. In particular, the entropy of posteriors is defined as $-\sum_{p_i \in \mathbf{y}} p_i \log p_i$, where p_i denotes the posterior for the i-th class. Figure 3.9 shows that maximal posterior achieves very high performance: In multiple datasets, we obtain AUC values above 0.8. Meanwhile, the AUC score is almost the same for all the three measures. This indicates that standard deviation and entropy can also be used as features for the attack.

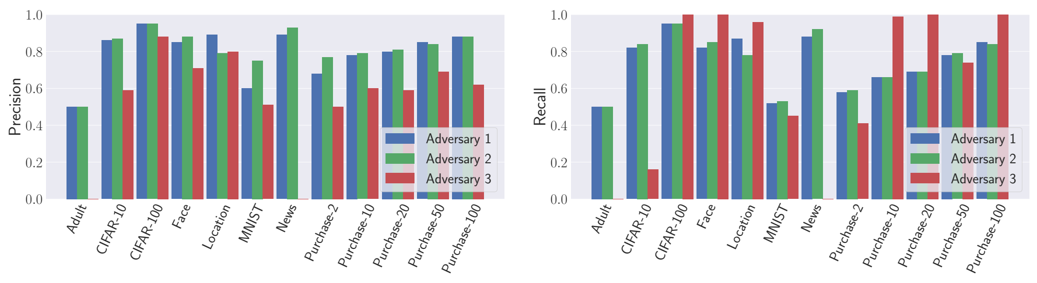
Next, we evaluate our concrete prediction following our threshold-choosing method. We generate 1,000 random data samples for each dataset and experiment multiple thresholds with respect to the top t percentile. Figure 3.10 shows the results. As we can see, setting t to 10 achieves a good performance (both precision and recall) for most of the datasets, such as CIFAR-100. Figure 3.11a further shows the maximal posterior distribution of member, non-member, and random samples for CIFAR-100. As the figure shows, our random samples' maximal posteriors behave similarly to the distribution of the non-member samples' which leads to the strong membership inference. On the other hand, our attack does not perform well on some datasets, such as Purchase-10, the corresponding maximal posteriors of which are shown in Figure 3.11b.

We also experiment with picking a fixed threshold for membership inference, e.g., maximal posterior above 50%. However, the evaluation shows that there is no single number that can achieve good performance for all the datasets. Thus, we conclude that our threshold-choosing method is suitable.



(a) CIFAR-100 **(b)** Purchase-10

Figure 3.11: The distribution of the maximal posterior of the ML model for the member, non-member, and random samples for two datasets: (a) CIFAR-100, (b) News. The y-axis shows the density.



(a) **(b)**

Figure 3.12: Comparison of the three different adversaries' performance. (a) precision, (b) recall.

3.4.4 Comparison of the Three Attacks

Figure 3.12 compares the performance, i.e., precision and recall, of the three attacks.[5] In particular, we show the best performance for our data transferring attack (adversary 2). As we can see, our first two adversaries achieve very similar performance for most of the datasets. On the other hand, the performance of our third adversary with the minimal assumptions is only slightly worse (especially for precision). These results clearly demonstrate that the membership inference attacks are very broadly applicable, thereby the corresponding risks are much more severe than previously shown.

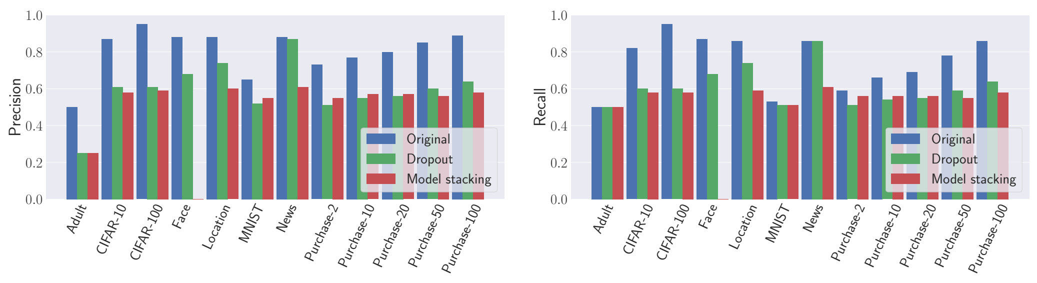
3.5 Defense

In this section, we propose two defense techniques aiming at mitigating membership privacy risks. The effectiveness of our membership inference attacks is mainly due to

[5]All the comparisons are done on the same dataset setting.

the overfitting nature of ML models. Therefore, our defense techniques are designed to increase ML models' generalizability, i.e., prevent them from being overfitted.

Our first technique is dropout which is designed for neural network-based classifiers.[6] Our second technique is model stacking. This mechanism is suitable for all ML models, independent of the classifier used to build them.



(a) (b)

Figure 3.13: Comparison of the first adversary's performance under both of the defense mechanisms. (a) precision, (b) recall.

As the first and second adversaries follow the same methodology of building a shadow model (with different assumptions on the dataset), we only show the effectiveness of our defense on the first adversary as well as on the third adversary to conserve space. For the first adversary, to fully assess the attack's performance under our defense, we further assume the attacker knows the defense technique being implemented and builds their shadow model following the same defense technique.

3.5.1 Dropout

Methodology. A fully connected neural network contains a large number of parameters which is prone to overfitting. Dropout is a very effective method to reduce overfitting based on empirical evidences. It is executed by randomly deleting in each training iteration a fixed proportion (dropout ratio) of edges in a fully connected neural network model. We apply dropout for both the input layer and the hidden layer (see Section 3.2) of the target model. We set our default dropout ratio to be 0.5.

Evaluation. We test dropout on all datasets against the first and third adversaries (except for the News and Adult datasets). Figure 3.13a and Figure 3.13b compare the performance of the first adversary's performance before and after the dropout defense. As we can see, the attack performance is reduced in almost all cases. For instance, the precision of the attack on the Purchase-100 dataset drops from 0.89 to 0.64, while the recall drops from 0.86 to 0.63. In another example, the precision and recall on the CIFAR-100 dataset drop by more than 30%. There is only one case where dropout does not help much, namely the target model trained on the News dataset.

Similarly, the performance of our third adversary is reduced due to dropout (see Figure 3.14). For example, the precision and recall of the attack on the CIFAR-100 dataset

[6]Note that for different kinds of classifiers, dropout can be replaced by other regulation techniques, such as the L_2-norm, which is also shown to be effective against the membership inference attack [75].

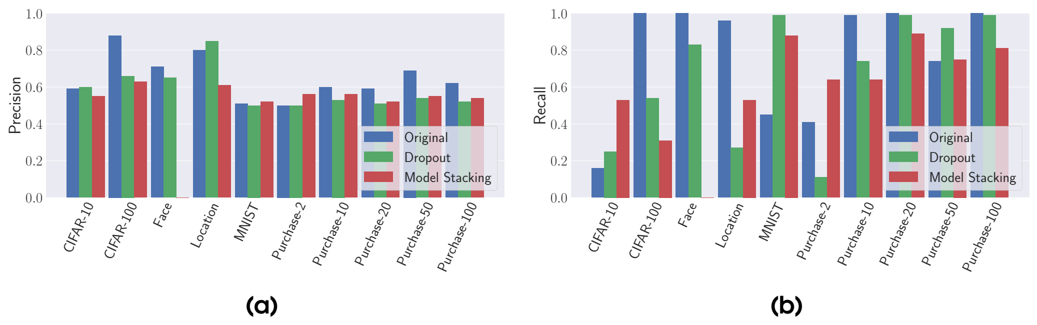


Figure 3.14: Comparison of the third adversary's performance under both defense mechanisms. (a) precision, (b) recall.

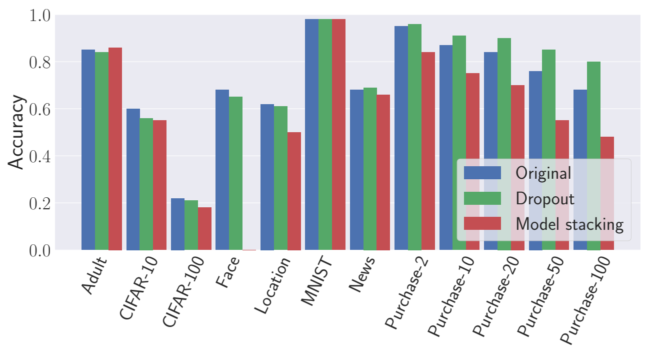


Figure 3.15: Comparison of the target model's accuracy under both defense mechanisms.

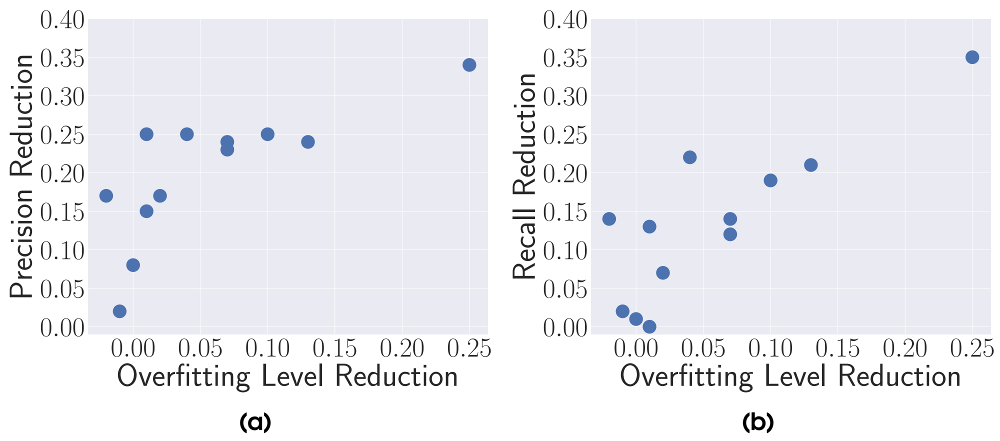
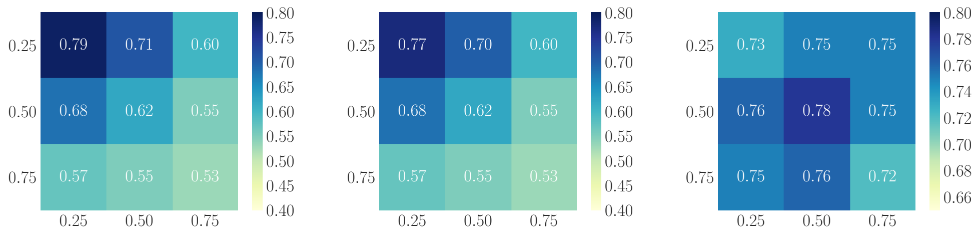


Figure 3.16: The relation between the overfitting level reduction (x-axis) and the first adversary's performance reduction (y-axis) when applying dropout as the defense mechanism. (a) precision reduction, (b) recall reduction.

drop by more than 25% and 40%, respectively. However, on some datasets, such as MNIST, the recall of the attack even improves. This indicates that our third adversary is more resistant to dropout than our first adversary.



(a) Precision **(b)** Recall **(c)** Accuracy

Figure 3.17: The effect of the dropout defense on the first adversary's performance, i.e., (a) precision and (b) recall, and on (c) the target model's accuracy under different dropout ratios in different layers of the neural network. The x-axis represents the input layer, and the y-axis represents the hidden layer.

Figure 3.15 further shows the original target model's performance (prediction accuracy) after dropout has been applied. We observe that, on more than half of the datasets, the dropout mechanism even increases the target model's prediction performance. For instance, on the Purchase-50 dataset, the target model's accuracy increases from 0.72 to 0.83.

Figure 3.16 plots the relation between the overfitting level (see Section 3.2) reduction and the first adversary's performance reduction after dropout has been applied. The overfitting level reduction is calculated as the original target model's overfitting level subtracting the dropout-defended target model's overfitting level. As we can see, more effective dropout which results in larger reduction on overfitting level leads to better defense against membership inference attacks. These results support the argument by Shokri et al. [75] that overfitting is a common enemy for the membership privacy risks and the target model's performance.

So far, we have used 0.5 as the dropout ratio. We further test the effect of varying the dropout ratio of our defense. We try different dropout ratios on both input and fully connected layers while monitoring the results on the first adversary's performance and the target model's accuracy. Figure 3.17 shows the result on the Purchase-100 dataset. We first observe that higher dropout ratio leads to lower attack performance. For instance, dropout ratio 0.75 on both layers reduces the attack's performance to 0.53 precision and recall. On the other hand, both large and small dropout ratio result in lower performance of the target model. This means the accuracy of the target model is the strongest when dropout ratio is mediate. In conclusion, 0.5 dropout ratio is a suitable choice for this defense technique.

3.5.2 Model Stacking

Methodology. The dropout technique is effective, however, it can only be applied when the target model is a neural network. To bypass this limitation, we present our second defense technique, namely model stacking, which works independently of the used ML classifier.

The intuition behind this defense is that if different parts of the target model are trained with different subsets of data, then the complete model should be less prone to overfitting. This can be achieved by using ensemble learning.

Ensemble learning is an ML paradigm, where instead of using a single ML model, multiple ML models are combined to construct the final model. There are different approaches to combine these ML models, such as bagging or boosting. For our defense, we focus on stacking the models in a hierarchical way. Figure 3.18 shows a sample architecture for model stacking.

Concretely, we organize the target model in two layers over three ML models. The first layer consists of two ML models (the first and second model). The second layer consists of a single ML model (the third model). As shown in Figure 3.18, to get the model's output on some data sample $\mathbf{x}$, we first apply $\mathbf{x}$ on each of the first two models to have their posteriors $\mathbf{y}^1$ and $\mathbf{y}^2$. We then concatenate both outputs, i.e., $\mathbf{y}^1||\mathbf{y}^2$, and apply the result to the third model which predicts the final output $\mathbf{y}$.

To maximize the prevention of overfitting, we train the three different models on disjoint sets of data. The intuition behind is that there is no data sample seen by more than one model during training.

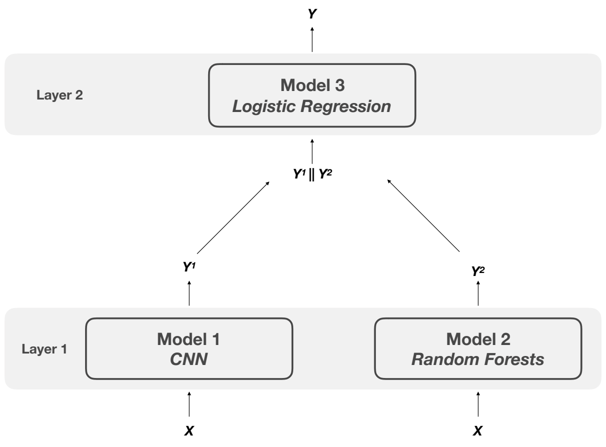


Figure 3.18: The architecture of model stacking.

Evaluation. For our evaluation, we use multilayer perceptron or CNN as the first model, random forests as the second model, and logistic regression as the third model. We pick this architecture to test the effect of using different machine learning models in the different layers. However, a different selection of models also suffices.

We build both target and shadow models for the first adversary as described, i.e., each model consists of 3 different ML models. To train our target and shadow models, we split the data into 12 disjoint sets. We use the first 6 sets to train and test our target model, and the remaining 6 to train and test the shadow model.

We evaluate this technique on all datasets but the Face dataset as it does not have enough data to provide meaningful results in this setting. Figure 3.13 shows the result for the first adversary. As we can see, model stacking reduces the attack's performance significantly in all cases. For instance, on the CIFAR-10 dataset, model stacking reduces the attack's precision and recall by more than 30%. Moreover, compared to the dropout defense, model stacking is more effective in some cases. Dropout does not

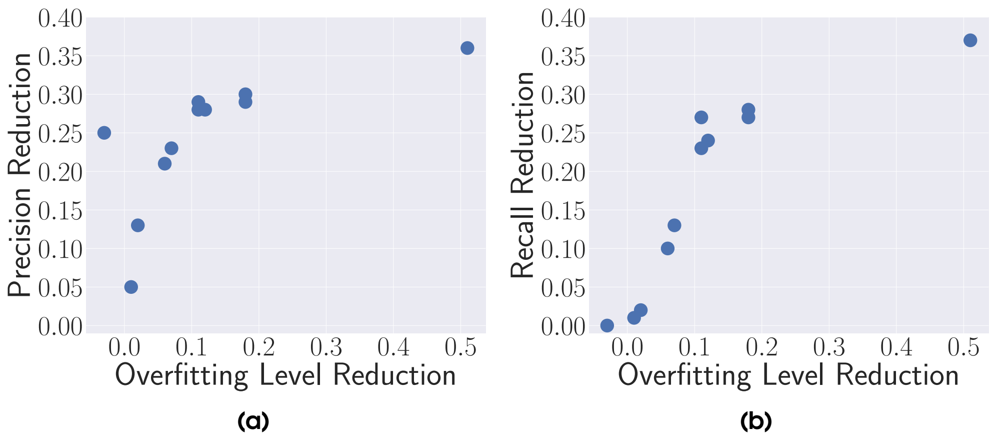


Figure 3.19: The relation between the overfitting level reduction (x-axis) and the first adversary's performance reduction (y-axis) when applying model stacking as the defense mechanism. (a) precision reduction, (b) recall reduction.

change the attack's performance on the News dataset while model stacking reduces the corresponding precision and recall by 28%. The same result can be observed on the Location dataset. However, model stacking affects target model's accuracy more than dropout in multiple cases, e.g., the Purchase datasets. The relation between the overfitting level reduction and attack performance reduction for the model stacking technique is very similar to the one for the dropout technique, the results are depicted in Figure 3.19.

Similarly, the performance of our third adversary drops after model stacking has been applied (see Figure 3.14). For instance, model stacking reduces the attack's performance on the Location dataset by more than 20% for the precision and 30% for the recall. But similar to the dropout defense, exceptions like MNIST also exist.

In conclusion, if the target model is not based on neural networks, model stacking is an effective defense technique. Otherwise, dropout is sufficient to mitigate the membership privacy risks due to its high utility maintenance.

3.6 Conclusion

Training data is a key factor that drives machine learning models being widely adopted in real-world applications. However, ML models suffer from membership privacy risks. The existing membership inference attacks have shown effective performance, but their applicability is limited due to strong assumptions on the threat model. In this chapter, we gradually relax these assumptions towards a more broadly applicable attack scenario.

Our **first adversary** utilizes only one shadow model. Extensive experiments show that this attack achieves a very similar performance as the previous one which utilizes multiple shadow models. As shadow models are established through MLaaS, our proposal notably reduces the cost of conducting the attack. We further perform the combining attack which does not require knowledge of the type of classifiers used in the target model.

The attack assumption is further relaxed for the **second adversary**, who does not have access to a dataset that comes from the same distribution as the training data of the target model. This is a more realistic attack scenario, but the previously proposed synthetic data generation solution can only be applied in specific cases. In contrast, we propose data transferring attacks, where the adversary utilizes another dataset to build a shadow model and generates the corresponding data to attack the target model. Through experiments, we have discovered that data transferring attack also achieves strong membership inference while being more general, realistic and widely applicable.

The **third adversary** has a minimal set of assumptions, i.e., they do not need to construct any shadow model and their attack is performed in an unsupervised way. We show that even in such a simple setting, membership inference is still effective.

Our evaluation is comprehensive and fully demonstrates the severe threat of membership privacy in ML models under those generalized conditions on 8 diverse datasets.

To remedy the situation, we propose **two defense mechanisms**. As we show the connection between overfitting and sensitivity to membership inference attacks, we investigate techniques that are designed to reduce overfitting. The first one, namely dropout, randomly deletes a certain proportion of edges in each training iteration in a fully connected neural network, while the second approach, namely model stacking, organizes multiple ML models in a hierarchical way. Extensive evaluation shows that indeed our defense techniques are able to largely reduce membership inference attack's performance, while maintaining a high-level utility, i.e., high target model's prediction accuracy.

4 Data Set Inference and Reconstruction Attacks

Updating Phase

4.1 Introduction

A key factor that drives the current ML development is the unprecedented large-scale data. In consequence, collecting high-quality data becomes essential for building advanced ML models. Data collection is a continuous process, which in turn transforms the ML model training into a continuous process as well: Instead of training an ML model for once and keeping on using it afterwards, the model's owner needs to keep on updating the model with newly-collected data. As training from scratch is often prohibitive, this is often achieved by *online learning.* In other words, the updating phase is introduced in the machine learning pipeline. We refer to the dataset used to perform model update as the *updating dataset.*

4.1.1 Our Contributions

In this chapter, we investigate the privacy of the updating phase. Updating a model results in its availability in two different versions, i.e., before and after being updated. In consequence, if the model is queried with the same set of data samples at two different points in time, it will provide different results. In this chapter, we investigate whether the change in the output of a black-box ML model before and after being updated can leak information of the dataset used to perform the update, namely the updating dataset. This constitutes a new attack surface against black-box ML models and such information leakage may compromise the intellectual property and data privacy of the ML model owner.

We concentrate on the most common ML application, i.e., classification. More importantly, we target black-box ML models; the most difficult attack setting where an adversary does not have access to their target model's parameters but can only query the model with their data samples and obtain the corresponding prediction results, i.e., *posteriors* in the case of classification.

In total, we propose four different attacks in this surface which can be categorized into two classes, namely, *single-sample attack class* and *multi-sample attack class.* The two attacks in the single-sample attack class concentrate on a simplified case when the target model is updated with one single data sample. We investigate this case to show whether an ML model's two versions' different outputs indeed constitute a valid attack surface. The two attacks in the multi-sample attack class tackle a more general and complex case when the updating dataset contains multiple data samples.

Among our four attacks, two (one for each attack class) aim at reconstructing the updating dataset which are the first attempts in this direction. Compared to many previous attacks inferring certain properties of a target model's training dataset [29, 39, 31], a dataset reconstruction attack leads to more severe consequences.

Our experiments show that indeed, the output difference of the same ML model's two different versions can be exploited to infer information about the updating dataset. We detail our contributions as the following.

General Attack Construction. Our four attacks follow a general structure, which can be formulated into an encoder-decoder style. The encoder realized by a multilayer perceptron (MLP) takes the difference of the target model's outputs, namely *posterior* *difference*, as its input while the decoder produces different types of information about the updating dataset with respect to different attacks.

To obtain the posterior difference, we randomly select a fixed set of data samples, namely *probing dataset*, and probe the target model's two different versions (before and after being updated with the updating dataset). Then, we calculate the difference between the two sets of posteriors as the input for our attack's encoder.

Single-Sample Attack Class. The single-sample attack class contains two attacks: *Single-sample label inference attack* and *single-sample reconstruction attack*. The first attack predicts the label of the single sample used to update the target model. We realize the corresponding decoder for the attack by a two-layer MLP. Our evaluation shows that our attack is able to achieve a strong performance, e.g., 0.96 accuracy on the CIFAR-10 [3] dataset.

The single-sample reconstruction attack aims at reconstructing the updating sample. To this end, we rely on autoencoders (AE). In detail, we first train an AE on a different set of data samples. Then, we transfer the AE's decoder into our attack model as its sample reconstructor. Experimental results show that we can reconstruct the single sample with a performance gain (with respect to mean squared error) of 22% for the MNIST dataset [2], 107.1% for the CIFAR-10 dataset, and 114.7% for the Insta-NY dataset [11], over randomly picking a sample affiliated with the same label of the updating sample.

Multi-Sample Attack Class. The multi-sample attack class includes *multi-sample label distribution estimation attack* and *multi-sample reconstruction attack*. Multi-sample label distribution estimation attack estimates the label distribution of the updating dataset's samples. It is a generalization of the label inference attack in the single-sample attack class. We realize this attack by setting up the attack model's decoder as a multilayer perceptron with a fully connected layer and a softmax layer. Kullback-Leibler divergence (KL-divergence) is adopted as the model's loss function. Our experiments demonstrate the effectiveness of this attack. For the CIFAR-10 dataset, when the updating dataset's cardinality is 100, our attack model achieves 0.00384 KL-divergence which outperforms random guessing by a factor of 2.5. Moreover, the accuracy of predicting the most frequent label is 0.29 which is almost 3 times higher than random guessing.

Our last attack, namely multi-sample reconstruction attack, aims at generating all samples in the updating dataset. This is a considerably more complex attack than the previous ones. The decoder for this attack is assembled with two components. The first one learns the data distribution of the updating dataset samples. In order to achieve coverage and accuracy of the reconstructed samples, we propose a novel hybrid generative model, namely CBM-GAN. Different from the standard generative adversarial networks (GANs), our Conditional Best of Many GAN (CBM-GAN) introduces a "Best Match" loss which ensures that *each* sample in the updating dataset is reconstructed *accurately*. The second component of our decoder relies on machine learning clustering to group the generated data samples by CBM-GAN into clusters and take the central sample of each cluster as one final reconstructed sample. Our evaluation shows that our approach outperforms all baselines when reconstructing the updating dataset on all MNIST, CIFAR-10, and Insta-NY datasets.

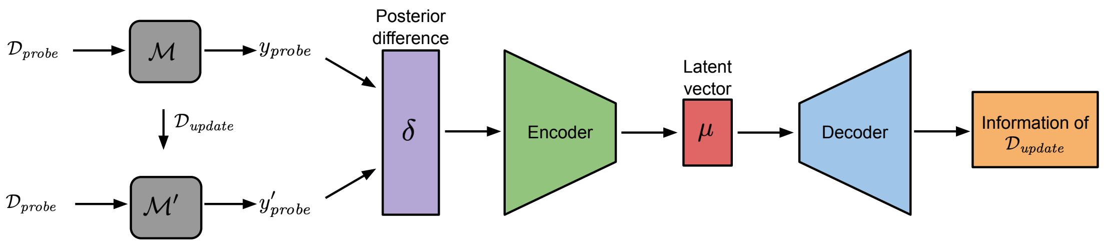


Figure 4.1: A schematic view of the general attack pipeline.

In summary, we make the following contributions in this chapter:

- We propose a new attack surface against black-box ML models in online learning (updating phase) scenarios, i.e., different outputs of the same ML model's two versions queried with the same set of data samples.
- We propose four different attacks in this surface based on advanced machine learning techniques. Extensive experiments demonstrate that the updating dataset's information can be effectively inferred.
- Two of our attacks aim at reconstructing the updating dataset itself. Though designed for our specific attack surface, we believe they can provide further insights into dataset reconstruction attacks in other settings.

4.1.2 Organization

The rest of the chapter is organized as follows. In Section 4.2, we introduce our general attack pipeline. Section 4.3 and Section 4.4 present our attacks and their evaluation results in single-sample and multi-sample attack classes, respectively. Section 4.5 performs further analysis on our attack including relaxing assumptions on attacker's knowledge. In Section 4.6, we discuss possible defense mechanisms. Finally, we conclude the chapter in Section 4.7.

4.2 General Attack Pipeline

Our general attack pipeline contains three phases. In the first phase, the adversary generates their attack input, i.e., posterior difference. In the second phase, our encoder transforms the posterior difference into a latent vector. In the last phase, the decoder decodes the latent vector to produce different information of the updating dataset with respect to our different attacks. Figure 4.1 provides a schematic view of our attack pipeline.

In this section, we first introduce our threat model, then we provide a general introduction for each phase of our attack pipeline. In the end, we present our strategy of deriving data to train our attack models.

4.2.1 Threat Model

For all of our four attacks, we consider an adversary with black-box access to the target model. This means that the adversary can only query the model with a set of data samples, i.e., their probing dataset, and obtain the corresponding posteriors. This is the most difficult attack setting for the adversary [75]. We also assume that the adversary has a local dataset which comes from the same distribution as the target model's training dataset following previous works [75, 31, P1]. Moreover, we consider the adversary to be able to establish the same ML model as the target model with respect to model architecture. This can be achieved by performing model hyperparameter stealing attacks [89, 60]. The adversary needs these two information to establish a shadow model which mimics the behavior of the target model to derive data for training their attack model (see Section 4.2). Also, part of the adversary's local dataset will be used as their probing dataset. Finally, we assume that the target model is updated only with new data, i.e., the updating dataset and the training dataset are disjoint.

We later show in Section 4.5 that the two assumptions, i.e., the adversary's knowledge of the target model's architecture and their possession of a dataset from the same distribution as the target model's training dataset, can be further relaxed.

4.2.2 Attack Input

Recall that we aim at investigating the information leaked from posterior difference of a model's two versions when queried with the same set of data samples. To create this posterior difference, the adversary first needs to pick a set of data samples as their probing dataset, denoted by $\mathcal{D}_{Probe}$. In this work, the adversary randomly selects the probing dataset ($\mathcal{D}_{Probe}$) from their local one. Choosing or crafting [60] a specific set of data samples as the probing dataset may further improve attack efficiency, we leave this as a future work. Next, the adversary queries the target model $\mathcal{M}$ with all samples in $\mathcal{D}_{Probe}$ and concatenates the received outputs to form a vector $\mathbf{y}_{probe}$. Then, they probe the updated model $\mathcal{M}'$ with samples in $\mathcal{D}_{Probe}$ and creates a vector $\mathbf{y}'_{probe}$ accordingly. In the end, they set the posterior difference, denoted by δ, to the difference of both outputs:

$$\delta = \mathbf{y}_{probe} - \mathbf{y}'_{probe}$$

Note that the dimension of δ is the product of $\mathcal{D}_{Probe}$'s cardinality and the number of classes of the target dataset. For instance, both CIFAR-10 and MNIST are 10-class datasets, while Insta-NY is an 8-class dataset. As our probing dataset always contains 100 data samples, this indicates the dimension of δ is 1,000 for CIFAR-10 and MNIST, and 800 for Insta-NY.

4.2.3 Encoder Design

All our attacks share the same encoder structure, we model it with a multilayer perceptron. The number of layers inside the encoder depends on the dimension of δ: Longer δ requires more layers in the encoder. As our δ is a 1,000-dimension vector for the MNIST and CIFAR-10 datasets, and 800-dimension vector for the Insta-NY dataset, we use two fully connected layers in the encoder. The first layer transforms

δ to a 128-dimension vector and the second layer further reduces the dimension to 64. The concrete architecture of our encoder is presented in Appendix A.1.2.

4.2.4 Decoder Structure

Our four attacks aim at inferring different information of $\mathcal{D}_{Update}$, ranging from samples' labels to the updating dataset itself. Thus, we construct the decoders of our attacks using different techniques. The details of these decoders will be presented in the following sections.

4.2.5 Shadow Model

Our encoder and decoder need to be trained jointly in a supervised manner. This indicates that we need ground truth data for model training. Due to our minimal assumptions, the adversary cannot get the ground truth from the target model. To solve this problem, we rely on shadow models following previous works [75, 31]. A shadow model is designed to mimic the target model. By controlling the training process of the shadow model, the adversary can derive the ground truth data needed to train their attack models.

As presented in Section 4.2.1, our adversary knows (1) the architecture of the target model and (2) a dataset coming from the same distribution as the target dataset. To build a shadow model $\mathcal{M}_{Shadow}$, the adversary first establishes an ML model with the same structure as the target model. Then, they get a shadow dataset $\mathcal{D}_{Shadow}$ from their local dataset (the rest is used as $\mathcal{D}_{Probe}$) and splits it into two parts: Shadow training dataset $\mathcal{D}^{Train}_{Shadow}$ and shadow updating dataset $\mathcal{D}^{Update}_{Shadow}$. $\mathcal{D}^{Train}_{Shadow}$ is used to train the shadow model while $\mathcal{D}^{Update}_{Shadow}$ is further split to m datasets: $\mathcal{D}^{Update^1}_{Shadow} \cdots \mathcal{D}^{Update^m}_{Shadow}$. The number of samples in each of the m datasets depends on the attack. For instance, our single-sample class attacks require each dataset containing a single sample. The adversary then generates m shadow updated models $\mathcal{M}'^1_{Shadow} \cdots \mathcal{M}'^m_{Shadow}$ by updating the shadow model $\mathcal{M}_{Shadow}$ with m shadow updating datasets in parallel.

The adversary, in the end, probes the shadow and updated shadow models with their probing dataset $\mathcal{D}_{probe}$, and calculates the shadow posterior difference $\delta^1_{Shadow} \cdots \delta^m_{Shadow}$. Together with the corresponding shadow updating dataset's ground truth information (depending on the attack), the training data for their attack model is derived.

More generally, the training dataset for each of our attack models contains m samples corresponding to $\mathcal{D}^{Update^1}_{Shadow} \cdots \mathcal{D}^{Update^m}_{Shadow}$. In all our experiments, we set m to 10,000. In addition, we create 1,000 updated models for the target model, this means the testing dataset for each attack model contains 1,000 samples, corresponding to $\mathcal{D}^{Update^1}_{Target} \cdots \mathcal{D}^{Update^{1,000}}_{Target}$.

4.3 Single-sample Attacks

In this section, we concentrate on the case when an ML model is updated with a single sample. This is a simplified attack scenario and we aim to examine the possibility of using posterior difference to infer information about the updating dataset. We start by introducing the single-sample label inference attack, then, present the single-sample reconstruction one.

4.3.1 Single-sample Label Inference Attack

Attack Definition. Our single-sample label inference attack takes the posterior difference as the input, and outputs the label of the single updating sample. More formally, given a posterior difference δ, our single-sample label inference attack is defined as follows:

$$\mathcal{A}_{LI} : \delta \mapsto \ell$$

where ℓ is a vector with each entry representing the probability of the updating sample affiliated with a certain label.

Methodology. To recap, the general construction of the attack model consists of an MLP-based encoder which takes the posterior difference as its input and outputs a latent vector μ. For this attack, the adversary constructs their decoder also with an MLP which is assembled with a fully connected layer and a softmax layer to transform the latent vector to the corresponding updating sample's label. The concrete architecture of our $\mathcal{A}_{LI}$'s decoder is presented in Appendix A.1.3.

To obtain the data for training $\mathcal{A}_{LI}$, the adversary generates ground truth data by creating a shadow model as introduced in Section 4.2, while setting the shadow updating dataset's cardinality to 1. Then, the adversary trains their attack model $\mathcal{A}_{LI}$ with a cross-entropy loss. Our loss function is,

$$\mathcal{L}_{CE} = \sum_i \ell_i \log(\hat{\ell}_i)$$

where ℓ_i is the true probability of label i and $\hat{\ell}_i$ is our predicted probability of label i. The optimization is performed by the ADAM optimizer.

To perform the label inference attack, the adversary constructs the posterior difference as introduced in Section 4.2, then feeds it to the attack model $\mathcal{A}_{LI}$ to obtain the label.

Experimental Setup. We evaluate the performance of our single-sample label inference attack using the MNIST, CIFAR-10, and Insta-NY datasets. First, we split each dataset into three disjoint datasets: The target dataset $\mathcal{D}_{Target}$, the shadow dataset $\mathcal{D}_{Shadow}$, and the probing dataset $\mathcal{D}_{Probe}$. As mentioned before, $\mathcal{D}_{Probe}$ contains 100 data samples. We then split $\mathcal{D}_{Shadow}$ to $\mathcal{D}_{Shadow}^{Train}$ and $\mathcal{D}_{Shadow}^{Update}$ to train the shadow model as well as updating it (see Section 4.2). The same process is applied to train and update the target model with $\mathcal{D}_{Target}$. As mentioned in Section 4.2, we build 10,000 and 1,000 updated models for the shadow and target models, respectively. This means the training and testing sets for our attack model contain 10,000 and 1,000 samples, respectively.

We use convolutional neural network (CNN) to build the shadow and target models for both CIFAR-10 and MNIST datasets, and a multilayer perceptron (MLP) for the Insta-NY dataset. The CIFAR-10 model consists of two convolutional layers, one max pooling layer, three fully connected layers, and a softmax layer. The MNIST model consists of two convolutional layers, two fully connected layers, and a softmax layer.

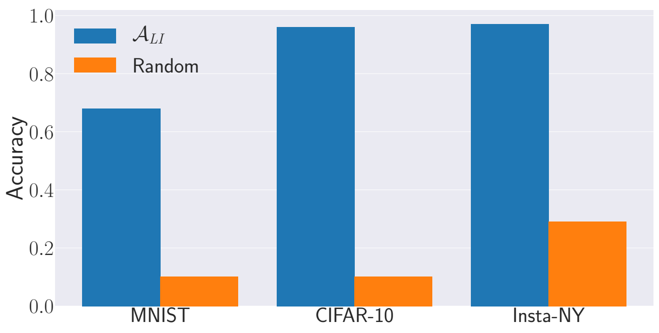


Figure 4.2: (Higher is better) Performance of the single-sample label inference attack ($\mathcal{A}_{LI}$) on MNIST, CIFAR-10, and Insta-NY datasets together with the baseline model. Accuracy is adopted as the evaluation metric.

Finally, the Insta-NY model consists of three fully connected layers and a softmax layer. The concrete architectures of the models are presented in Appendix A.1.1.

All shadow and target models' training sets contain 10,000 images for CIFAR-10 and MNIST, and 5,000 samples for Insta-NY. We train the CIFAR-10, MNIST and Insta-NY models for 50, 25, and 25 epochs, respectively, with a batch size of 64. To create an updated ML model, we perform a single-epoch training. Finally, we adopt accuracy to measure the performance of the attack. All of our experiments are implemented using Pytorch [7]. For reproducibility purposes, our code is available at `https://github.com/AhmedSalem2/Updates-Leak`.

Results. Figure 4.2 depicts the experimental results. As we can see, $\mathcal{A}_{LI}$ achieves a strong performance with an accuracy of 0.97 on the Insta-NY dataset, 0.96 on the CIFAR-10 dataset, and 0.68 on the MNIST dataset. Moreover, our attack significantly outperforms the baseline model, namely Random, which simply guesses a label over all possible labels. As both CIFAR-10 and MNIST contain 10 balanced classes, the baseline model's result is approximately 10%. For the Insta-NY dataset, since it is not balanced, we randomly sample a label for each sample to calculate the baseline, which results in approximately 29% accuracy. Our evaluation shows that the different outputs of an ML model's two versions can indeed leak information of the corresponding updating dataset.

4.3.2 Single-sample Reconstruction Attack

Attack Definition. Our single-sample reconstruction attack takes one step further to construct the data sample used to update the model. Formally, given a posterior difference δ, our single-sample reconstruction attack, denoted by $\mathcal{A}_{SSR}$, is defined as follows:

$$\mathcal{A}_{SSR} : \delta \mapsto \mathbf{x}_{Update}$$

where $\mathbf{x}_{Update}$ denotes the sample used to update the model ($\mathcal{D}_{Update} = \{\mathbf{x}_{Update}\}$).

Methodology. Reconstructing a data sample is a much more complex task than predicting the sample's label. To tackle this problem, we need an ML model which is able to generate data samples. To this end, we rely on an autoencoder (AE).

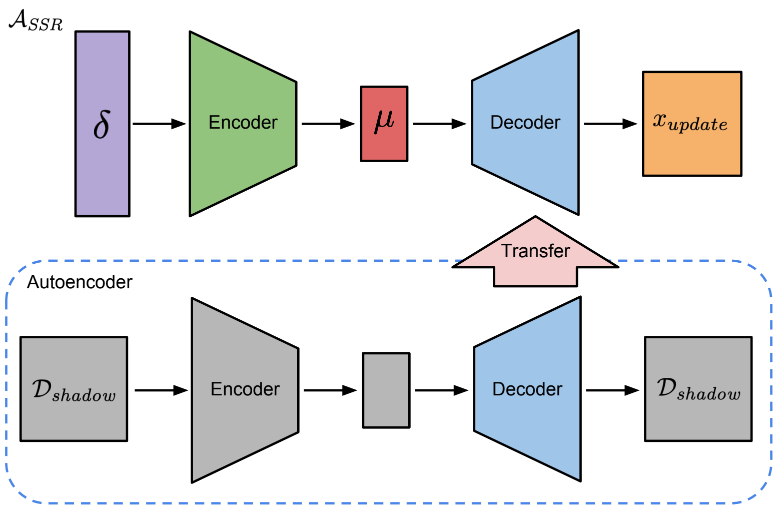


Figure 4.3: Methodology of the single-sample reconstruction attack ($\mathcal{A}_{SSR}$).

Autoencoder is assembled with an encoder and a decoder. Different from our attacks, AE's goal is to learn an efficient encoding for a data sample: Its encoder encodes a sample into a latent vector and its decoder tries to decode the latent vector to reconstruct the same sample. This indicates that the AE's decoder itself is a data sample reconstructor. For our attack, we first train an AE, then transfer the AE's decoder to our attack model as the initialization of the attack's decoder. Figure 4.3 provides an overview of the attack methodology. The concrete architectures of our AEs' encoders and decoders are presented in Appendix A.1.4.

After the autoencoder is trained, the adversary takes its decoder and appends it to their attack model's encoder. To establish the link, the adversary adds an additional fully connected layer to its encoder which transforms the dimensions of the latent vector μ to the same dimension as μ_{AE}.

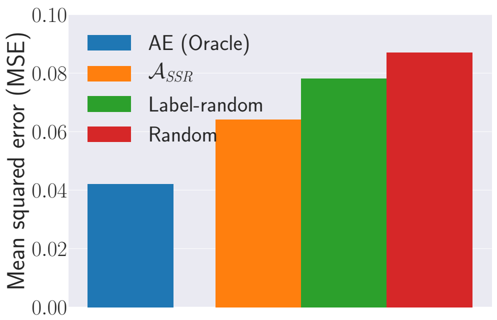
We divide the attack model training process into two phases. In the first phase, the adversary uses their shadow dataset to train an AE with the previously mentioned model architecture. In the second phase, they follow the same procedure for single-sample label inference attack to train their attack model. Note that the decoder from AE here serves as the initialization of the decoder, this means it will be further trained together with the attack model's encoder. To train both the autoencoder and our attack model, we use mean squared error (MSE) as the loss function. Our objective is,

$$\mathcal{L}_{MSE} = \|\hat{\mathbf{x}}_{Update} - \mathbf{x}_{Update}\|_2^2$$

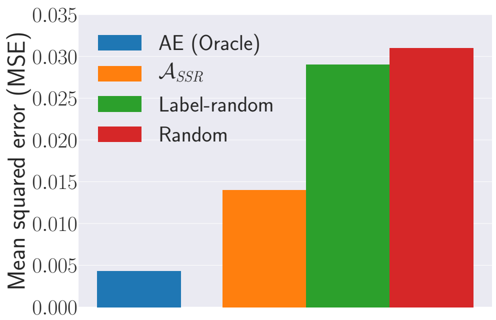
where $\hat{\mathbf{x}}_{Update}$ is our predicted data sample. We again adopt ADAM as the optimizer.

Experimental Setup. We use the same experimental setup as the previous attack (see Section 4.3.1) except for the evaluation metric. In detail, we adopt MSE to measure our attack's performance instead of accuracy.

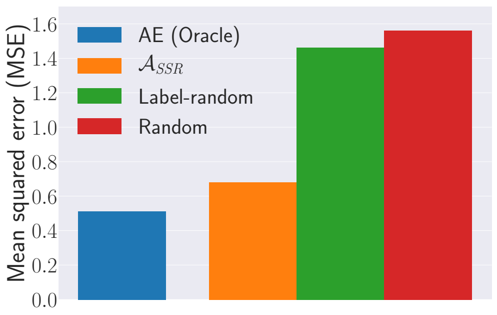
We construct two baseline models, namely Label-random and Random. Both of these baseline models take a random data sample from the adversary's shadow dataset.



(a) MNIST



(b) CIFAR-10



(c) Insta-NY

Figure 4.4: (Lower is better) Performance of the single-sample reconstruction attack ($\mathcal{A}_{SSR}$) together with autoencoder and two baseline models. Mean squared error is adopted as the evaluation metric. Autoencoder (AE) serves as an oracle as the adversary cannot use it for their attack.

The difference is that the Label-random baseline picks a sample within the same class as the target updating sample, while the Random baseline takes a random data sample from the whole shadow dataset of the adversary. The Label-random baseline can be implemented by first performing our single-sample label inference attack to learn the label of the data sample and then picking a random sample affiliated with the same label.

Results. First, our single-sample reconstruction attack achieves a promising performance. As shown in Figure 4.4, our attack on the MNIST dataset outperforms the Random baseline by 36% and more importantly, outperforms the Label-random baseline by 22%. Similarly, for the CIFAR-10 and Insta-NY datasets, our attack achieves an MSE of 0.014 and 0.68 which is significantly better than the two baseline models, i.e., it outperforms the Label-random (Random) baselines by a factor of 2.1 (2.2) and 2.1 (2.3), respectively. The difference between our attack's performance gain over the baseline models on the MNIST and on the other datasets is expected; as the MNIST dataset is more homogeneous compared to the other two. In other words, the chance of picking a random data sample similar to the updating sample is much higher in the MNIST dataset than in the other datasets.

Secondly, we compare our attack's performance against the results of the autoencoder for sample reconstruction. Note that AE takes the original data sample as input and

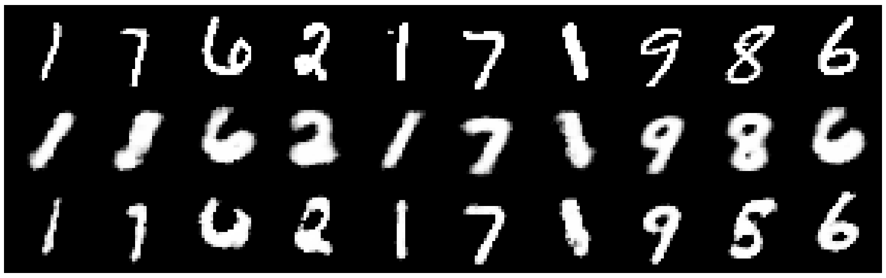


Figure 4.5: Visualization of some generated samples from the single-sample reconstruction attack ($\mathcal{A}_{SSR}$) on the MNIST dataset. Samples are fair random draws, not cherry-picked. The first row shows the original samples. The second row shows the reconstructed samples by $\mathcal{A}_{SSR}$. The third shows row the reconstructed samples by autoencoder, i.e., the upper bound of our reconstruction attack.

outputs the reconstructed one, thus it is considered as an oracle, since the adversary does not have access to the original updating sample. Here, we just use AE's result to show the best possible result for our attack. From Figure 4.4, we observe that AE achieves 0.042, 0.0043, and 0.51 MSE for the MNIST, CIFAR-10, and Insta-NY datasets, respectively, which indeed outperforms our attack. However, our attack still has a comparable performance.

Finally, Figure 4.5 visualizes some randomly sampled reconstructed images by our attack on MNIST. The first row depicts the original images used to update the models and the second row shows the result of our attack. As we can see, our attack is able to reconstruct images that are visually similar to the original sample with respect to rotation and shape. We also show the result of AE in the third row in Figure 4.5 which as mentioned before, is the upper bound for our attack. The results from Figure 4.4 and Figure 4.5 demonstrate the strong performance of our attack.

4.4 Multi-sample Attacks

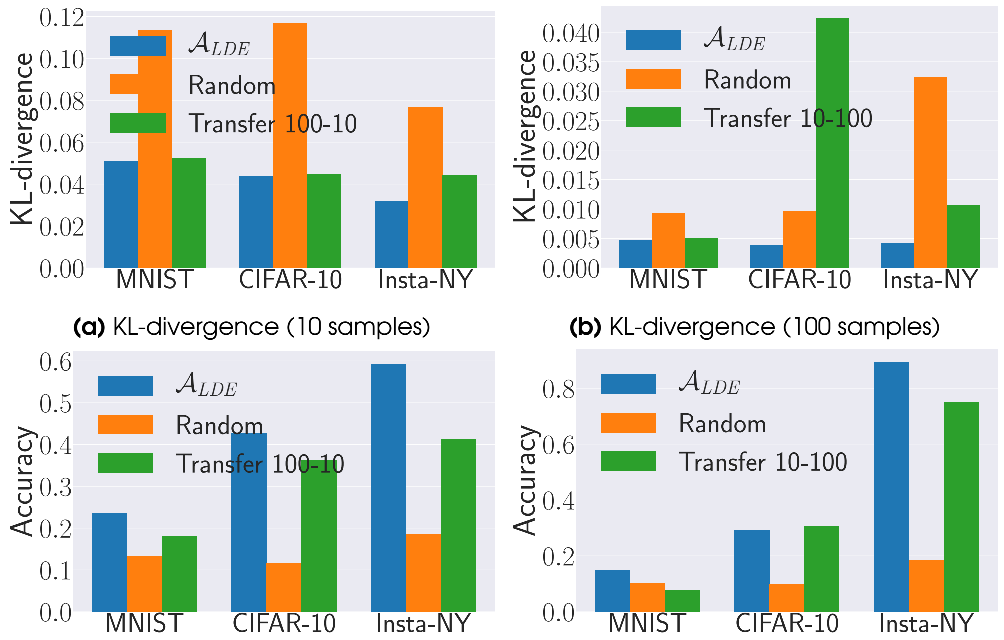
After demonstrating the effectiveness of our attacks against the updating dataset with a single sample, we now focus on a more general attack scenario where the updating dataset contains multiple data samples that are never seen during the training. We introduce two attacks in the multi-sample attack class: Multi-sample label distribution estimation attack and multi-sample reconstruction one.

4.4.1 Multi-sample Label Distribution Estimation Attack

Attack Definition. Our first attack in the multi-label attack class aims at estimating the label distribution of the updating dataset's samples. It can be considered as a generalization of the label inference attack in the single-sample attack class. Formally, the attack is defined as:

$$\mathcal{A}_{LDE} : \delta \mapsto \mathbf{q}$$

where $\mathbf{q}$ denotes a vector representing the distribution of labels over all classes for samples in the updating dataset.



(a) KL-divergence (10 samples) **(b)** KL-divergence (100 samples)

(c) Accuracy (10 samples) **(d)** Accuracy (100 samples)

Figure 4.6: (Lower is better for (a) and (b), higher is better for (c) and (d)) Performance of the multi-sample label distribution estimation attack ($\mathcal{A}_{LDE}$) together with the baseline model and transfer attack. KL-divergence and accuracy are adopted as the evaluation metric. Accuracy here is used to measure the prediction of the most frequent label over samples in the updating dataset. Transfer 10-100 means that each of the training and testing samples for the attack model corresponds to an updating dataset containing 10 and 100 data samples, respectively.

Methodology. The adversary uses the same encoder structure as presented in Section 4.2 and the same decoder structure of the label inference attack (Section 4.3.1). Since the label distribution estimation attack estimates a probability vector $\mathbf{q}$ instead of performing classification, we use Kullback-–Leibler divergence (KL-divergence) as our objective function:

$$\mathcal{L}_{KL} = \sum_i (\hat{\mathbf{q}}_\ell)_i \log \frac{(\hat{\mathbf{q}}_\ell)_i}{(\mathbf{q}_\ell)_i}$$

where $\hat{\mathbf{q}}_\ell$ and $\mathbf{q}_\ell$ represent our attack's estimated label distribution and the target label distribution, respectively.

To train the attack model $\mathcal{A}_{LDE}$, the adversary first generates their training data as mentioned in Section 4.2. They then train $\mathcal{A}_{LDE}$ with the posterior difference $\delta^1_{Shadow} \cdots \delta^m_{Shadow}$ as the input and the normalized label distribution of their corresponding updating datasets as the output. We assume the adversary knows the cardinality of the updating dataset. We try to relax this assumption later in our evaluation.

Experimental Setup. We evaluate our label distribution estimation attack using updating dataset of cardinalities 10 and 100. For the two different cardinalities, we build attack models as mentioned in the methodology. All data samples in each updating dataset for the shadow and target models are sampled uniformly, thus each sample (in both training and testing dataset) for the attack model, which corresponds to an updating dataset, has the same label distribution of the original dataset. We use a batch size of 64 when updating the models.

For evaluation metrics, we calculate KL-divergence for each testing sample (corresponding to an updating dataset on the target model) and report the average result over all testing samples (1,000 in total). Besides, we also measure the accuracy of predicting the most frequent label over samples in the updating dataset. We randomly sample a dataset with the same size as the updating dataset and use its samples' label distribution as the baseline, namely Random.

Results. We report the result for our label distribution estimation attack in Figure 4.6. As shown, $\mathcal{A}_{LDE}$ achieves a significantly better performance than the Random baseline on all datasets. For the updating dataset with 100 data samples on the CIFAR-10 dataset, our attack achieves 3 and 2.5 times better accuracy and KL-divergence, respectively, than the Random baseline. Similarly, for the MNIST and Insta-NY datasets, our attack achieves 1.5 and 4.8 times better accuracy, and 2 and 7.9 times better KL-divergence. Furthermore, $\mathcal{A}_{LDE}$ achieves a similar improvement over the Random baseline for the updating dataset of size 10.

Recall that the adversary is assumed to know the cardinality of the updating dataset in order to train their attack model, we further test whether we can relax this assumption. To this end, we first update the shadow model with 100 samples, while updating the target model with only 10. As shown in Figure 4.6a and Figure 4.6c, our attack still has a similar performance as the original attack. However, when the adversary updates their shadow model with 10 data samples, while the target model is updated with 100 ones (Figure 4.6b and Figure 4.6d), our attack performance drops significantly, in particular for KL-divergence on the CIFAR-10 dataset. We believe this is due to the 10 samples not providing enough information for the attack model to generalize to a larger updating dataset.

4.4.2 Multi-sample Reconstruction Attack

Attack Definition. Our last attack, namely multi-sample reconstruction attack, aims at reconstructing the updating dataset. This attack can be considered as a generalization of the single-sample reconstruction attack, and a step towards the goal of reconstructing the training dataset of a black-box ML model. Formally, the attack is defined as follows:

$$\mathcal{A}_{MSR} : \delta \mapsto \mathcal{D}_{Update}$$

where $\mathcal{D}_{Update} = \{\mathbf{x}^1_{Update}, \dots, \mathbf{x}^{|\mathcal{D}_{Update}|}_{Update}\}$ contains the samples used to update the model.

Methodology. The complexity of the task for reconstructing an updating dataset increases significantly when the updating dataset size grows from one to multiple. Our single-sample reconstruction attack (Section 4.3.2) uses an AE to reconstruct a single sample. However, AE cannot generate a set of samples. In fact, directly predicting a set of examples is a very challenging task. Therefore, we rely on generative models which are able to generate multiple samples rather than a single one.

We first introduce the classical Generative Adversarial Networks (GANs) and point out why classical GANs cannot be used for our multi-sample reconstruction attack. Next, we propose our Conditional Best of Many GAN (CBM-GAN), a novel hybrid generative model and demonstrate how to use it to execute the multi-sample reconstruction attack.

Generative Adversarial Networks. Samples from a dataset are essentially samples drawn from a complex data distribution. Thus, one way to reconstruct the dataset $\mathcal{D}_{Update}$ is to learn this complex data distribution and sample from it. This is the approach we adopt for our multi-sample reconstruction attack. Mainly, the adversary starts the attack by learning the data distribution of $\mathcal{D}_{Update}$, then they generate multiple samples from the learned distribution, which is equivalent to reconstructing the dataset $\mathcal{D}_{Update}$. In this work, we leverage the state-of-the-art generative model GANs, which has been demonstrated effective on learning a complex data distribution.

A GAN consists of a pair of ML models: a generator (G) and a discriminator (D). The generator G learns to transform a Gaussian noise vector $z \sim \mathcal{N}(0,1)$ to a data sample $\hat{x}$,

$$\mathrm{G} : z \mapsto \hat{x}$$

such that the generated sample $\hat{x}$ is indistinguishable from a true data sample. This is enabled by the discriminator D which is jointly trained. The generator G tries to fool the discriminator, which is trained to distinguish between samples from the Generator (G) and true data samples. The objective function maximized by the GAN's discriminator D is,

$$\mathcal{L}_D = \mathbb{E}_{x \in \mathcal{D}_{Update}} \log(\mathrm{D}(x)) + \mathbb{E}_{\hat{x}} \log(1 - \mathrm{D}(\hat{x})) \tag{4.1}$$

The GAN discriminator D is trained to output 1 ("true") for real data and 0 ("false") for fake data. On the other hand, the generator G maximizes:

$$\mathcal{L}_G = \mathbb{E}_{\hat{x}} \log(\mathrm{D}(\hat{x}))$$

Thus, G is trained to produce samples $\hat{x} = \mathrm{G}(z)$ that are classified as "true" (real) by D.

However, our attack aims to reconstruct $\mathcal{D}_{Update}$ for any given δ, which the standard GAN does not support. Therefore, first, we change the GAN into a conditional model to condition its generated samples $\hat{x}$ on the posterior difference δ. Second, we construct our novel hybrid generative model CBM-GAN, by adding a new "Best Match" loss to reconstruct *all* samples inside the updating dataset *accurately.*

CBM-GAN. The decoder of our attack model is casted as our CBM-GAN's generator (G). To enable this, we concatenate the noise vector z and the latent vector μ produced by our attack model's encoder (with posterior difference as input), and use it as CBM-GAN's generator's input, as in Conditional GANs [56]. This allows our decoder to map the posterior difference δ to samples in $\mathcal{D}_{Update}$.

However, Conditional GANs are severely prone to mode collapse, where the generator's output is restricted to a limited subset of the distribution [19, 92]. To deal with this, we introduce a reconstruction loss. This reconstruction loss forces our GAN to cover all the modes of the distribution (set) of data samples used to update the model. However, it is unclear, given a posterior difference δ and a noise vector z pair, which

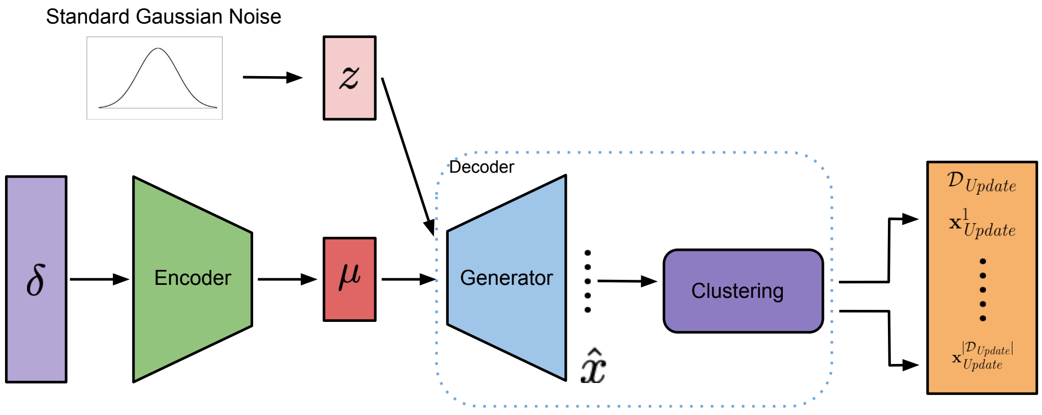


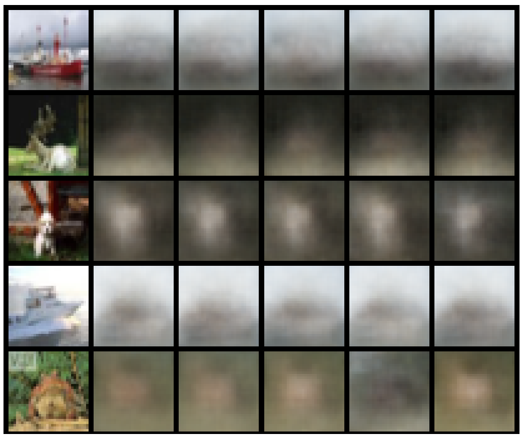
Figure 4.7: Methodology of the multi-sample reconstruction attack ($\mathcal{A}_{MSR}$).

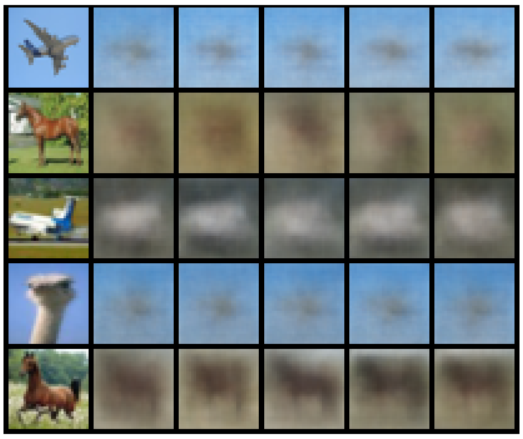
sample in the data distribution we should force CBM-GAN to reconstruct. Therefore, we allow our GAN full flexibility in learning a mapping from posterior difference and noise vector z pairs to data samples – this means we allow it to choose the data sample to reconstruct. We realize this using a novel "Best Match" based objective in the CBM-GAN formulation,

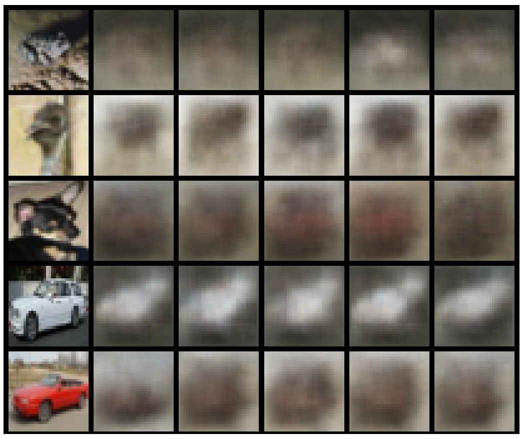
$$\mathcal{L}_{BM} = \sum_{x \in \mathcal{D}_{Update}} \min_{\hat{x} \sim \mathrm{G}} \|\hat{x} - x\|_2^2 + \sum_{\hat{x}} \log(\mathrm{D}(\hat{x})) \tag{4.2}$$

where $\hat{x} \sim \mathrm{G}$ represents samples produced by our CBM-GAN given a latent vector μ and noise sample z. The first part of the $\mathcal{L}_{BM}$ objective is based on the standard MSE reconstruction loss and forces our CBM-GAN to reconstruct all samples in $\mathcal{D}_{Update}$ as the error is summed across $x \in \mathcal{D}_{Update}$. However, unlike the standard MSE loss, given a data sample $x \in \mathcal{D}_{Update}$, the loss is based only on the generated sample $\hat{x}$ which is closest to the data sample $x \in \mathcal{D}_{Update}$. This allows CBM-GAN to reconstruct samples in $\mathcal{D}_{Update}$ without having an explicit mapping from posterior difference and noise vector z pairs to data samples, as only the "Best Match" is penalized. Finally, the discriminator D ensures that the samples $\hat{x}$ are indistinguishable from the "true" samples of $\mathcal{D}_{Update}$.

Figure 4.7 presents a schematic view of our multi-sample reconstruction attack's methodology. The concrete architecture of CBM-GAN's generator and discriminator for the three datasets used in this chapter are listed in Appendix A.1.5. *Training of CBM-GAN.* The training of the attack model $\mathcal{A}_{MSR}$ is more complicated than previous attacks, hence we provide more details here. Similar to the previous attacks, the adversary starts the training by generating the training data as mentioned in Section 4.2. They then jointly train their encoder and CBM-GAN with the posterior difference $\delta^1_{Shadow} \cdots \delta^m_{Shadow}$ as the inputs and samples inside their corresponding updating datasets,


(a)


(b)


(c)

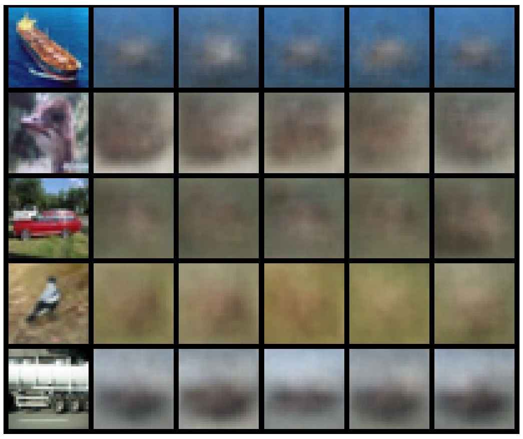

(d)

Figure 4.8: Visualization of some generated samples from the multi-sample reconstruction attack ($\mathcal{A}_{MSR}$) before clustering on the CIFAR-10 dataset. Samples are fair random draws, not cherry-picked. The left column shows the original samples and the next 5 columns show the 5 nearest reconstructed samples with respect to mean squared error.

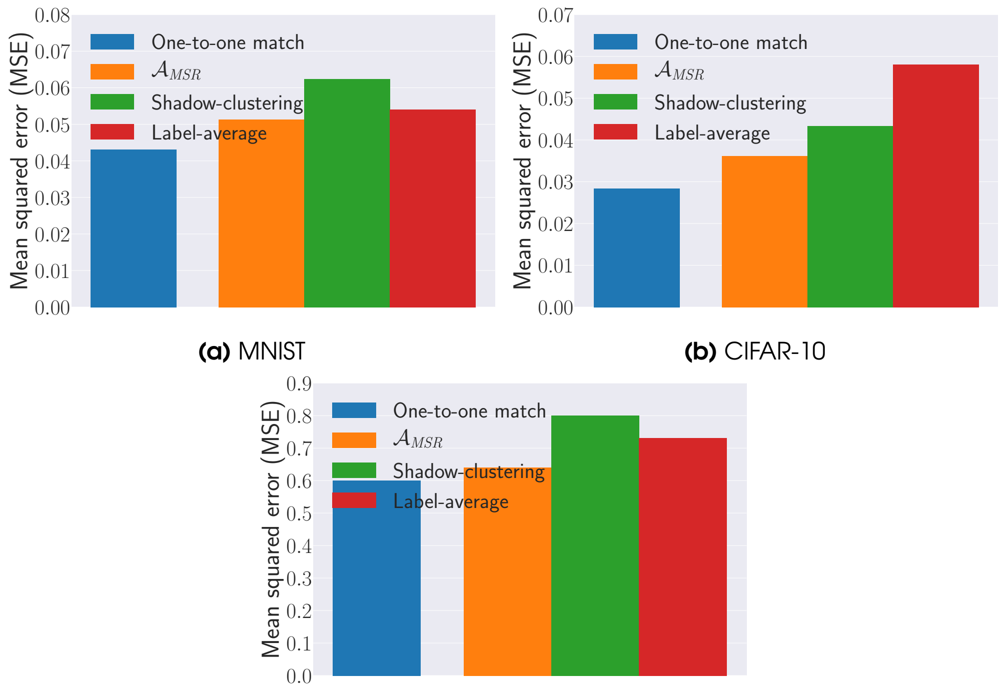
i.e., $\mathcal{D}_{Shadow}^{Update^1} \cdots \mathcal{D}_{Shadow}^{Update^m}$, as the output. More concretely, for each posterior difference δ_{Shadow}^{i}, they update their attack model $\mathcal{A}_{MSR}$ as follows:

1. The adversary sends the posterior difference δ_{Shadow}^{i} to their encoder to get the latent vector μ_i.

2. They then generate $|\mathcal{D}_{Shadow}^{Update^i}|$ noise vectors.

3. To create the generator's input, they concatenate each of the noise vectors with the latent vector μ_i.

4. On the input of the concatenated vectors, the CBM-GAN generates $|\mathcal{D}_{Shadow}^{Update^i}|$ samples, i.e., each vector corresponds to one sample.



(a) MNIST **(b)** CIFAR-10

(c) Insta-NY

Figure 4.9: (Lower is better) Performance of the multi-sample reconstruction attack ($\mathcal{A}_{MSR}$) together with one-to-one match and the two baseline models. Mean squared error (MSE) is adopted as the evaluation metric. The match between the original and reconstructed samples is performed by the Hungarian algorithm for both $\mathcal{A}_{MSR}$ and Shadow-clustering. For Label-average, each sample is matched within the average of samples with the same class in the shadow dataset. One-to-one match serves as an oracle as the adversary cannot use it for their attack.

5. The adversary then calculates the generator loss as introduced by Equation 4.2, and uses it to update the generator and the encoder.

6. Finally, they evaluate and update the CBM-GAN's discriminator according to Equation 4.1.

Clustering. CBM-GAN only provides a generator which learns the distribution of the samples in the updating dataset. However, to reconstruct the exact data samples in $\mathcal{D}_{Update}$, we need a final step assisted by machine learning clustering. In detail, we assume the adversary knows the cardinality of $\mathcal{D}_{Update}$ as in Section 4.4.1. After the CBM-GAN is trained, the adversary utilizes CBM-GAN's generator to generate a large number of samples. They then cluster the generated samples into $|\mathcal{D}_{Update}|$ clusters. Here, the K-means algorithm is adopted to perform clustering where we set K to $|\mathcal{D}_{Update}|$. In the end, for each cluster, the adversary calculates its centroid, and takes the nearest sample to the centroid as one reconstructed sample.

Experimental Setup. We evaluate the multi-sample reconstruction attack on the updating dataset of size 100 and generate 20,000 samples for each updating dataset reconstruction with CBM-GAN. For the rest of the experimental settings, we follow the one mentioned in Section 4.4.1 except for evaluation metrics and baseline.

We use MSE between the updating and reconstructed data samples to measure the multi-sample reconstruction attack's performance. We construct two baselines, namely Shadow-clustering and Label-average. For Shadow-clustering, we perform K-means clustering on the adversary's shadow dataset. More concretely, we cluster the adversary's shadow dataset into 100 clusters and take the nearest sample to the centroid of each cluster as one reconstructed sample. For Label-average, we calculate the MSE between each sample in the updating dataset and the average of the images with the same label in the adversary's shadow dataset.

Results. In Figure 4.8, we first present some visualization of the intermediate result of our attack, i.e., the CBM-GAN's output before clustering, on the CIFAR-10 dataset. For each randomly sampled image in the updating dataset, we show the 5 nearest reconstructed images with respect to the MSE. As we can see, our attack model tries to generate images with similar characteristics to the original ones. For instance, the 5 reconstructed images for the airplane image in Figure 4.8b show a blue background and a blurry version of the airplane itself. Similar result can be observed from the boat image in Figure 4.8a, the car image in Figure 4.8c, and the boat image in Figure 4.8d. It is also interesting to see that CBM-GAN provides different samples for the two different horse images in Figure 4.8b. The blurriness in the results is expected, due to the complex nature of the CIFAR-10 dataset and the weak assumptions for our adversary, i.e., access to black-box ML model.

We also quantitatively measure the performance of our intermediate results, by calculating the MSE between each image in the updating dataset and its nearest reconstructed sample. We refer to this as one-to-one match. Figure 4.9 shows for the CIFAR-10, MNIST, and Insta-NY datasets, we achieve 0.0283, 0.043 and 0.60 MSE, respectively. It is important to note that the adversary cannot perform one-to-one match as they does not have access to ground truth samples in the updating dataset, i.e., one-to-one match is an oracle.

Figure 4.9 shows the mean squared error of our full attack with clustering for all datasets. To match each of our reconstructed samples to a sample in $\mathcal{D}_{Update}$, we rely on the Hungarian algorithm [45]. This guarantees that each reconstructed sample is only matched with one ground truth sample in $\mathcal{D}_{Update}$ and vice versa. As we can see, our attack outperforms both baseline models on the CIFAR-10, MNIST and Insta-NY datasets (20%, 22%, and 25% performance gain for Shadow-clustering and 60.1%, 5.5% and 14% performance gain for Label-average, respectively). The different performance gain of our attack over the label-average baseline for different datasets is due to the different complexity of these datasets. For instance, all images inside MNIST have black background and lower variance within each class compared to the CIFAR-10 dataset. The different complexity results in some datasets having a more representative label-average, which leads to a lower performance gain of our attack over them.

These results show that our multi-sample reconstruction attack provides a more useful output than calculating the average from the adversary's dataset. In detail, our attack achieves an MSE of 0.036 on the CIFAR-10 dataset, 0.051 on the MNIST dataset, and 0.64 on the Insta-NY dataset. As expected, the MSE of our final attack is higher

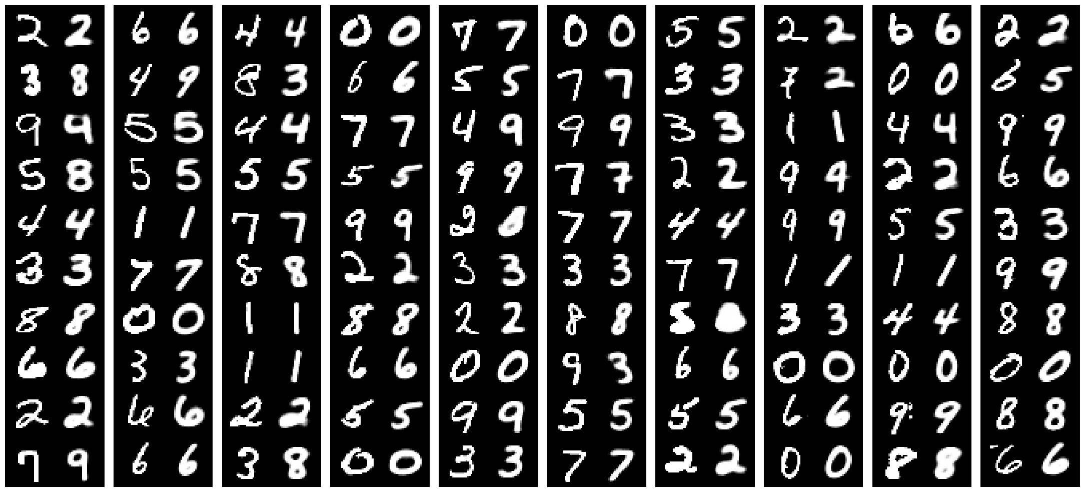


Figure 4.10: Visualization of a full MNIST updating dataset together with the output of the multi-sample reconstruction attack ($\mathcal{A}_{MSR}$) after clustering. Samples are fair random draws, not cherry-picked. The left column shows the original samples and the right column shows the reconstructed samples. The match between the original and reconstructed samples is performed by the Hungarian algorithm.

than one-to-one match, i.e., the above mentioned intermediate results.

We further visualize our full attack's result on the MNIST dataset. Figure 4.10 shows a sample of a full MNIST updating dataset reconstruction, i.e., the CBM-GAN's reconstructed images for the 100 original images in the updating dataset. We observe that our attack model reconstructs diverse digits of each class that for most of the cases match the actual ground truth data very well. This suggests CBM-GAN is able to capture most modes in a data distribution well. Moreover, comparing the results of this attack (Figure 4.10) with the results of the single-sample reconstruction attack (Figure 4.5), we can see that this attack produces sharper images. This result is due to the discriminator of our CBM-GAN, as it is responsible for making the CBM-GAN's output to look real, i.e., sharper in this case.

One limitation of our attack is that CBM-GAN's sample generation and clustering are performed separately. In the future, we plan to combine them to perform an end-to-end training which may further boost our attack's performance.

From all these results, we show that our attack does not generate a general representation of data samples affiliated with the same label, but tries to reconstruct images with similar characteristics as the images inside the updating dataset (as shown by the different shapes of the same numbers in Figure 4.10).

Finally, although our two reconstruction attacks (Section 4.4.2 and Section 4.3.2) are designed specifically for the online learning setting, we believe the underlying techniques we propose, i.e., pretrained decoder from a standard autoencoder and CBM-GAN, can be further extended to reconstruct datasets from black-box ML models in other settings.

Relaxing The Knowledge of Updating Dataset Cardinality. One of the above attack's main assumptions is the adversary's knowledge of the updating dataset cardinality, i.e., $|\mathcal{D}_{Update}|$. Next, we show how to relax this assumption. To recap, the

Table 4.1: Evaluation of the data transferability attacks. The first column shows all different attacks, the second and third shows the performance of the attacks using similar and different distributions, respectively. Where $\mathcal{A}_{LI}$ performance is measured in accuracy, $\mathcal{A}_{MSR}$ and $\mathcal{A}_{SSR}$ measured in MSE, and $\mathcal{A}_{LDE}(10)$ and $\mathcal{A}_{LDE}(100)$ measured in accuracy (KL-divergence).

Attack	Original	Transfer
$\mathcal{A}_{LI}$	0.97	0.89
$\mathcal{A}_{SSR}$	0.68	1.1
$\mathcal{A}_{LDE}(10)$	0.59(0.0317)	0.55(0.0377)
$\mathcal{A}_{LDE}(100)$	0.89(0.0041)	0.89 (0.0067)
$\mathcal{A}_{MSR}$	0.64	0.73

adversary needs the updating dataset cardinality when updating their shadow model and clustering CBM-GAN's output. We address the former by using updating datasets of different cardinalities. For the latter, we use the silhouette score to find the optimal k for K-means, i.e., the most likely value of the target updating dataset's cardinality. The silhouette score lies in the range between -1 and 1, it reflects the consistency of the clustering. Higher silhouette score leads to more suitable k.

Specifically, the adversary follows the previously presented methodology in Section 4.4.2 with the following modifications. First, instead of using updating datasets with the same cardinality, the adversary uses updating datasets with different cardinalities to update the shadow model. Second, after the adversary generates multiple samples from CBM-GAN, they uses the silhouette score to find the optimal k. The silhouette score is used here to identify the target model's updating dataset cardinality from the different updating datasets cardinalities used to update the shadow model.

We evaluate the effectiveness of this attack on all datasets. We use a target model updated with 100 samples and create our shadow updated models using updating datasets with cardinality 10 and 100. Concretely, we update the shadow model half of the time with updating datasets of cardinality 10 and the other half with cardinality 100.

Our evaluation shows that our attack consistently produces higher silhouette score -by at least 20%- for the correct cardinality in all cases. In other words, our method can always detect the right cardinality of the updating dataset in this setting. Moreover, the MSE for the final output of the attack only drops by 1.6%, 0.8%, and 5.6% for the Insta-NY, MNIST, and CIFAR-10 datasets, respectively.

4.5 Discussion

In this section, we analyze the effect of different hyperparameters of both the target and shadow models on our attacks' performance. Furthermore, we investigate relaxing the threat model assumptions and discuss the limitations of our attacks.

Relaxing The Attacker Model Assumption. Our threat model has two main assumptions: Same data distribution for both target and shadow datasets and same structure for both target and shadow models. We relax the former by proposing data transferability attack and latter by model transferability attack.

Data Transferability. In this setting, we locally train and update the shadow model with a dataset which comes from a different distribution compared to the target dataset. For our experiments, we use Insta-NY as the target dataset and Insta-LA as the shadow dataset.

Table 4.1 depicts the evaluation results. As expected, the performance of our data transferability attacks drops; however, they are still significantly better than corresponding baseline models. For instance, the performance of the multi-sample reconstruction attack drops by 14%, but is still 10% better than the baseline (see Figure 4.9). Moreover, the multi-sample label distribution attack's accuracy (KL-divergence) only drops by 6.8% (18.9%) and 0% (63%), which is still significantly better than the baseline (see Figure 4.6) by 6.5x (2x) and 4.6x (4.8x) for updating dataset sizes of 10 and 100, respectively.

Model Transferablity. Now we relax the attacker's knowledge on the target model's architecture, i.e., we use different architectures for shadow and target models. In our experiments on Insta-NY, we use the same architecture mentioned previously in Section 4.3.1 for the target model; we then remove one hidden layer and use half of the number of neurons in other hidden layers for the shadow model.

The performance drop of our model transferability attack is only less than 2% for all of our attacks, which shows that our attacks are robust against such changes in the model architectures. We observe similar results when repeating the experiment using different architectures.

Effect of The Probing Dataset Cardinality. We evaluate the performance of our attacks on CIFAR-10 when the probing dataset cardinality is 10, 100, 1,000, or 10,000. As our encoder's input size relies on the probing dataset cardinality (see Section 4.2), we adjust its input layer size accordingly.

As expected, using a probing dataset of size 10 reduces the performance of the attacks. For instance, the single-sample label inference and reconstruction attacks' performance drops by 9% and 71%, respectively. However, increasing the probing dataset cardinality from 100 to 1,000 or 10,000 has a limited effect (up to 3.5% performance gain). It is also important to mention that the computational requirement for our attacks increases with an increasing probing dataset cardinality, as the cardinality decides the size of the input layer for our attack models. In conclusion, using 100 samples for probing the target model is a suitable choice.

Effect of Target Model Hyperparameters. We now evaluate our attacks' performance with respect to two hyperparameters of the target model.

Target Model's Training Epochs Before Updating. We use the MNIST dataset to evaluate the multi-sample label distribution estimation attack's performance on target models trained for 10, 20, and 50 epochs. For each setting, we update the model and execute our attack as mentioned in Section 4.4.1.

The experiments show that the difference in the attack's performance for the different models is less than 2%. That is expected as gradients are not monotonically decreasing during the training procedure. In other words, information is not necessarily

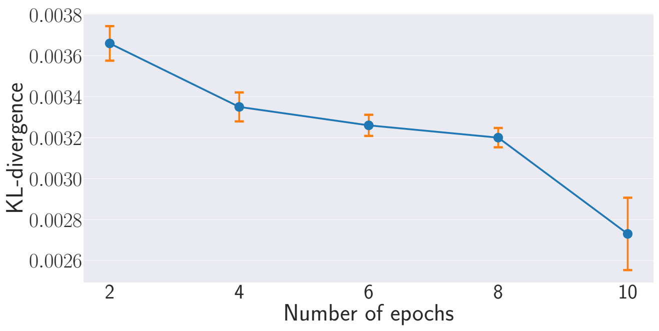


Figure 4.11: (Lower is better) The performance of the multi-sample label distribution estimation attack ($\mathcal{A}_{LDE}$) with different number of epochs used to update the target model.

vanishing [34].

Target Model's Updating Epochs. We train target and shadow models as introduced in Section 4.4.1 with the Insta-NY dataset, but we update the models using different number of epochs. More concretely, we update the models using from 2 to 10 epochs and evaluate the multi-sample label distribution estimation attack's performance on the updated models.

We report the results of our experiments in Figure 4.11. As expected, the multi-sample label distribution estimation attack's performance improves with the increase of the number of epochs used to update the model. For instance, the attack performance improves by 25.4 % when increasing the number of epochs used to update the model from 2 to 10.

Limitations of Our Attacks. For all of our attacks, we assume a simplified setting, in which, the target model is solely updated on new data. Moreover, we perform our attacks on updating datasets of maximum cardinality of 100. In future work, we plan to further investigate a more complex setting, where the target model is updated using larger updating datasets of both new and old data.

4.6 Possible Defenses

Adding Noise to Posteriors. All our attacks leverage posterior difference as the input. Therefore, to reduce our attacks' performance, one could sanitize posterior difference. However, the model owner cannot directly manipulate the posterior difference, as they do not know with what or when the adversary probes their model. Therefore, they have to add noise to the posterior for each queried sample independently. We have tried adding noise sampled from a uniform distribution to the posteriors. Experimental results show that the performance for some of our attacks indeed drops to a certain degree. For instance, the single-sample label inference attack on the CIFAR-10 dataset drops by 17% in accuracy. However, the performance of our multi-sample reconstruction attack stays stable. One reason might be the noise vector z is part of CBM-GAN's input which makes the attack model more robust to the noisy input.

Differential Privacy. Another possible defense mechanism against our attacks is differentially private learning. Differential privacy [26] can help an ML model learn its main tasks while reducing its memory on the training data. If differentially private learning schemes [8, 74, 23] are used when updating the target model, this by design will reduce the performance of our attacks. However, it is also important to mention that depending on the privacy budget for differential privacy, the utility of the model can drop significantly.

We leave an in-depth exploration of effective defense mechanisms against our attacks as a future work.

4.7 Conclusion

Large-scale data being generated at every second turns ML model training into a continuous process. In consequence, a machine learning model queried with the same set of data samples at two different time points will provide different results. In this chapter, we investigate whether these different model outputs can constitute a new attack surface for an adversary to infer information of the dataset used to update the model. We propose four different attacks in this surface all of which follow a general encoder-decoder structure. The encoder encodes the difference in the target model's output before and after being updated, and the decoder generates different types of information regarding the updating dataset.

We start by exploring a simplified case when an ML model is only updated with one single data sample. We propose two different attacks for this setting. The first attack shows that the label of the single updating sample can be effectively inferred. The second attack utilizes an autoencoder's decoder as the attack model's pretrained decoder for single-sample reconstruction.

We then generalize our attacks to the case when the updating dataset contains multiple samples. Our multi-sample label distribution estimation attack trained using a KL-divergence loss is able to infer the label distribution of the updating dataset effectively. For the multi-sample reconstruction attack, we propose a novel hybrid generative model, namely CBM-GAN, which uses a "Best Match" loss in its objective function. The "Best Match" loss directs CBM-GAN's generator to reconstruct each sample in the updating dataset. Quantitative and qualitative results show that our attacks achieve promising performance.

5 Model Hijacking Attack

Training Phase

5.1 Introduction

The demand for computational resources and high-quality data to train deep machine learning models is significantly increasing. Consequently, individual users face multiple challenges in satisfying the needed requirements for reaching state-of-the-art performance. Instead, new training paradigms which involve multiple parties jointly building machine learning models have been proposed, e.g., one such training paradigm is federated learning [16]. However, this inclusion of new parties in the training process of ML models raises new security and privacy risks.

More concretely, this inclusion of new parties creates a new attack surface where an adversary can manipulate the training of an ML model, i.e., the *training phase.* This type of attacks is called training time attack. Some examples in this domain include backdoor [36] and data poisoning attacks [13].

5.1.1 Our Contributions

In this chapter, 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 possible risk of 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. 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[1].

To perform the model hijacking attack, the adversary only needs the ability to

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

poison the target model's 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 when both the original and hijacking datasets are 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 to, 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 ones. 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 one. To hijack a target model, the Chameleon attack poisons the original dataset using the camouflaged dataset. Then, 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.

To demonstrate the efficacy of model hijacking, we perform experiments in different settings using three benchmark computer vision datasets including MNIST [2],CIFAR-10 [3], and CelebA [51]. 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 attack achieves 58.6% and 73.7% accuracy with a negligible drop in performance for their original tasks, respectively.

Abstractly, this chapter's contributions can be summarized as:

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

2. We propose the Camouflager model/architecture 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.

5.1.2 Organization

The rest of this chapter is organized as follows. We introduce the model hijacking attack and our two attacks, i.e., the Chameleon and Adverse Chameleon attacks, in Section 5.2. Next, we evaluate the performance of our two different attacks in Section 5.3 and discuss their limitations in Section 5.4. Finally, we conclude the chapter in Section 5.5.

5.2 Model Hijacking Attack

In this section, we present our different techniques for the model hijacking attack. First, we introduce the problem statement of our attack, then its threat model. Next, we present the general pipeline of the model hijacking attack, and clarify the difference between it 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.

5.2.1 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 a 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 5.2.3) 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.

5.2.2 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 [80, 43, 73]. Moreover, we assume the adversary has a hijackee dataset which they rely on to create the camouflaged dataset from the hijacking one. In Section 5.2.7, 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 for camouflaging 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 [84, 13, 43, 15, 80, 97, 72].

5.2.3 General Attack Pipeline

Table 5.1: The description of the different datasets used in the model hijacking attack.

Name	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 like 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.

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 the 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*.

The Camouflager is trained using both a hijacking and a *hijackee* datasets. As mentioned in Section 5.2.2, samples in the hijackee dataset are visually similar to samples in the original one. The Camouflager is trained using two types of losses, namely *Visual Loss* and *Semantic Loss*. The Visual Loss makes the Camouflager's output, i.e., the camouflaged dataset, visually similar to the hijackee one; while the Semantic Loss makes the camouflaged dataset semantically similar to the hijacking one, 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 5.1.

5.2.4 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 threat model 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 [43, 79], the adversary tries to jeopardize the models' utility, i.e., increasing the misclassification rate, by manipulating the training of the target model. Similarly, backdoor attacks [36, 49] can also have the same aim. In addition to that, in the backdoor attack, the adversary can link a trigger with a 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. Moreover, 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 model hijacking attack with the only restriction of having similar or fewer labels compared to the target model's original task.

5.2.5 Building Blocks

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

Camouflager (*Cam*): 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 5.1. As the figure shows, the Camouflager consists of two encoders and one decoder. The first encoder ($\mathcal{E}_O$) takes a sample ($\mathbf{x}_O$) from the hijackee dataset as its input, while the other encoder ($\mathcal{E}_H$) takes a sample ($\mathbf{x}_H$) 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 $\mathbf{x}_C$ which has visual appearance of $\mathbf{x}_O$, but with the features/semantics of $\mathbf{x}_H$. More formally,

$$Cam(\mathbf{x}_O, \mathbf{x}_H) = \mathcal{E}^{-1}\Big(\mathcal{E}_O(\mathbf{x}_O) || \mathcal{E}_H(\mathbf{x}_H)\Big) = \mathbf{x}_C,$$

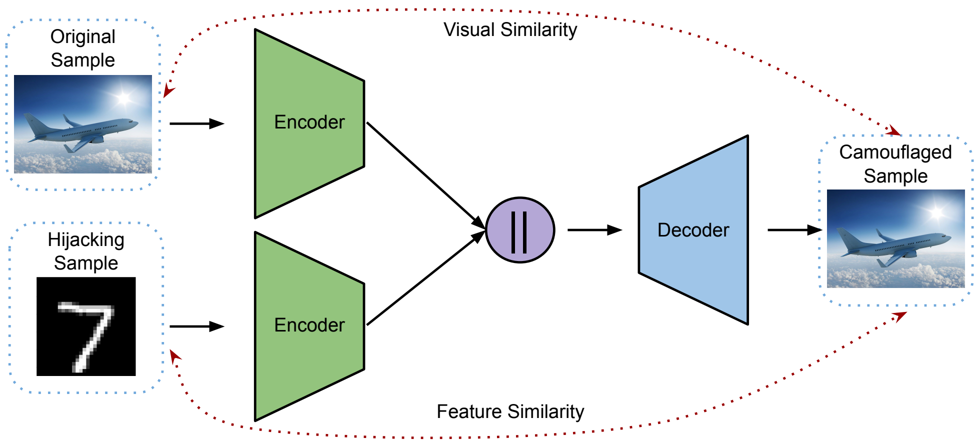


Figure 5.1: A schematic view of the Camouflager. The encoders first encode the original and the hijacking samples. Then the outputs of the encoders are concatenated and inputted to the decoder. The decoder then generates the camouflaged sample which has the visual appearance of the original sample but the features of the hijacking one.

where $\mathbf{x}_O$ denotes a sample from the original dataset ($\mathbf{x}_O \in \mathcal{D}_O$), $\mathbf{x}_H$ a sample from the hijacking dataset ($\mathbf{x}_H \in \mathcal{D}_H$), and $\mathbf{x}_C$ the camouflaged sample.

Visual Loss ($\mathcal{L}_{vl}$): Next, we introduce the first loss of the Camouflager, i.e., the Visual Loss. This loss drives the Camouflager to output data that has a 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:

$$\mathcal{L}_{vl} = \min \|\mathbf{x}_C - \mathbf{x}_O\|$$

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 one, 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 5.2.2, we use a pretrained MobileNetV2 [71] 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:

$$\mathcal{L}_{sl} = \min \|\mathcal{F}(\mathbf{x}_C) - \mathcal{F}(\mathbf{x}_H)\|,$$

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 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:

$$\mathcal{L}_{asl} = \max \|\mathcal{F}(\mathbf{x}_C) - \mathcal{F}(\mathbf{x}_O)\|,$$

5.2.6 The Chameleon Attack

After presenting the general pipeline and 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 sets up their hijacking and hijackee datasets. To recap, this hijackee dataset is used for camouflaging the hijacking one.

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 5.2.5), 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 5.1, 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.

$$\mathcal{L}_{Cham}(\mathbf{x}_C, \mathbf{x}_O, \mathbf{x}_H) = \min\Big(\|\mathbf{x}_C - \mathbf{x}_O\| + \|\mathcal{F}(\mathbf{x}_C) - \mathcal{F}(\mathbf{x}_H)\|\Big) \tag{5.1}$$

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.

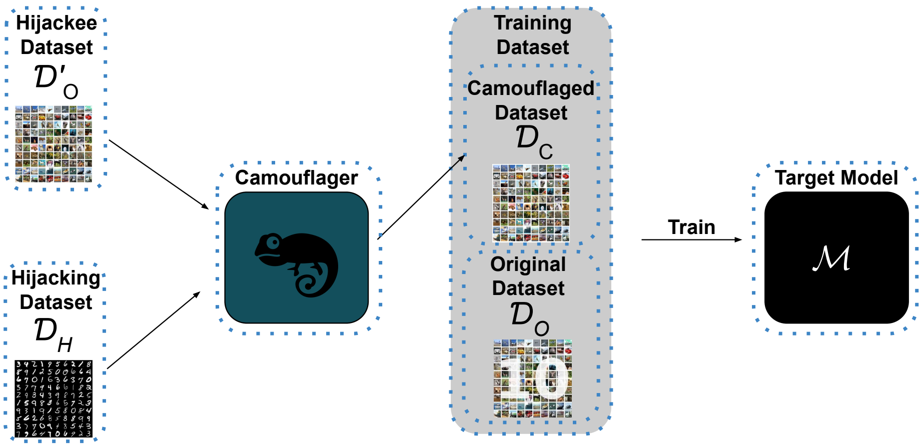


Figure 5.2: An overview of the model hijacking attack. First, the adversary inputs the hijackee and hijacking datasets to the Camouflager. Next, they take the Camouflager's output (the camouflaged dataset) and poison the training dataset of the target model. Finally, the model is trained with the poisoned dataset.

For the Chameleon attack, the adversary uses the hijackee dataset and the hijacking dataset to train the Camouflager as follows:

1. For each epoch, the adversary first randomly pairs samples from the hijackee dataset to the samples in the hijacking one. 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, the adversary feeds each pair (a hijackee and a hijacking samples) to the Camouflager. Next, the Camouflager's output and the input samples are used to update the Camouflager using the loss function introduced in Equation 5.1.

Executing: After training the Camouflager, the adversary can now execute their attack. Figure 5.2 shows an overview of the Chameleon attack after the training of the Camouflager. As the figure shows, first, the adversary maps the samples inside the hijackee dataset to samples from the hijacking one. They then create 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 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.

5.2.7 The Adverse Chameleon Attack

As will be shown later in Section 5.3, 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 a 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 5.1 as the loss for training the Camouflager, we use the following loss.

$$\begin{aligned}\mathcal{L}_{ChamAdv}(\mathbf{x}_C, \mathbf{x}_O, \mathbf{x}_H) = \min \Big(\|\mathbf{x}_C - \mathbf{x}_O\| + \|\mathcal{F}(\mathbf{x}_C) - \mathcal{F}(\mathbf{x}_H)\| \\ - \|\mathcal{F}(\mathbf{x}_C) - \mathcal{F}(\mathbf{x}_O)\| \Big)\end{aligned} \tag{5.2}$$

Besides the different loss function, to execute the Adverse Chameleon attack, the adversary follows the same steps as the Chameleon attack (Section 5.2.7).

5.3 Evaluation

In this section, we present our experimental results. We start by introducing our 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.

5.3.1 Evaluation Settings

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

5.3.1.1 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 [38]. The Resnet18 model expects inputs with the size 224×224; hence we resize our datasets, i.e., MNIST, CIFAR-10, and CelebA, accordingly.

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 [71].

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:

$$\begin{aligned}\mathbf{x}_{In} \rightarrow\ & \texttt{Conv2d(4,12)}\\ & \texttt{Conv2d(4,24)}\\ & \texttt{Conv2d(4,48)}\\ & \texttt{Conv2d(4,96)} \rightarrow \mu\end{aligned}$$

Here, $\mathbf{x}_{In}$ 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:

$$\begin{aligned}(\mu_O||\mu_H) \rightarrow\ & \texttt{ConvTranspose2d(4,96)}\\ & \texttt{ConvTranspose2d(4,48)}\\ & \texttt{ConvTranspose2d(4,24)}\\ & \texttt{ConvTranspose2d(4,3)} \rightarrow \mathbf{x}_{Out}\end{aligned}$$

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 $\mathbf{x}_{Out}$ 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).

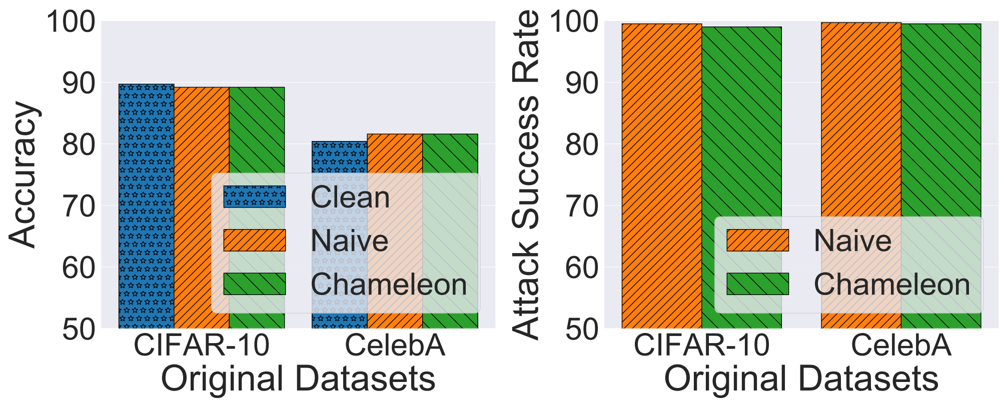
Finally, we use the Adam optimizer to train the Camouflager.

5.3.1.2 Evaluation Metrics

To evaluate the performance of our model hijacking attack, we use 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, one 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.

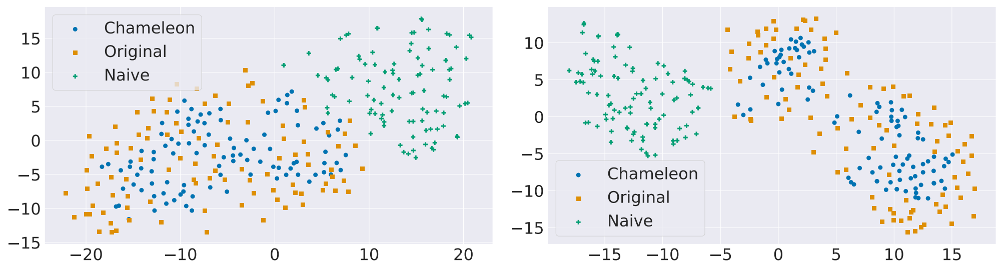
Attack Success Rate: The Attack Success Rate measures the model hijacked attack performance on the hijacking dataset. We calculate the Attack Success Rate by computing the accuracy of the hijacked model on a hijacking testing dataset, i.e., a testing dataset with the same distribution as the hijacking 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.



(a) Utility

(b) Attack Success Rate

Figure 5.3: 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 5.3a compares the Utility of both the Naive and Chameleon attacks with a clean model (Clean) using the original testing dataset, and Figure 5.3b compares the Attack Success Rate of both attacks on the hijacking testing dataset.



(a) CIFAR-10

(b) CelebA

Figure 5.4: 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 5.4a for the CIFAR-10 dataset, and Figure 5.4b for the CelebA dataset. Here MNIST is the hijacking dataset, and CIFAR-10 (Figure 5.4a) and CelebA (Figure 5.4b) are the original ones.

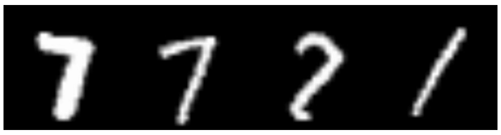
5.3.2 The Chameleon Attack

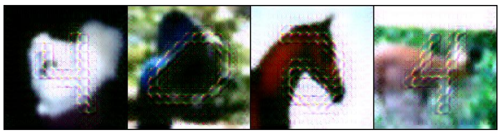
After introducing the datasets and our evaluations metrics, we now evaluate the performance of our Chameleon attack. We use MNIST as our hijacking dataset and both CIFAR-10 and CelebA as the original ones for this attack.

Firstly, we map the labels of the hijacking dataset and the original dataset as mentioned in Section 5.2.6. Next, we train the Camouflager by constructing two encoders and a decoder with the architectures presented in Section 5.3.1.1, 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 5.2.5 to train the Camouflager.

(a) Original CIFAR-10

(b) Original CelebA


(c) Original MNIST


(d) Camouflaged CIFAR-10

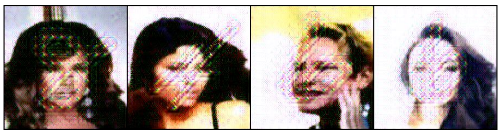

(e) Camouflaged CelebA

Figure 5.5: Visualization of the output of the Camouflager for the Chameleon Attack for both the CIFAR-10 (Figure 5.5d) and CelebA (Figure 5.5e) datasets. Moreover, we show samples for both the Original(Figure 5.5a and Figure 5.5b) and hijacking (Figure 5.5c) datasets.

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).

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 5.3a. As the figure shows, our Chameleon attack achieves almost the same Utility, as both the Clean and Naive models have a similar performance. 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 [T3].

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 5.3b 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 5.6).

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) [54] 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 5.4a and Figure 5.4b, respectively. As Figure 5.4 clearly

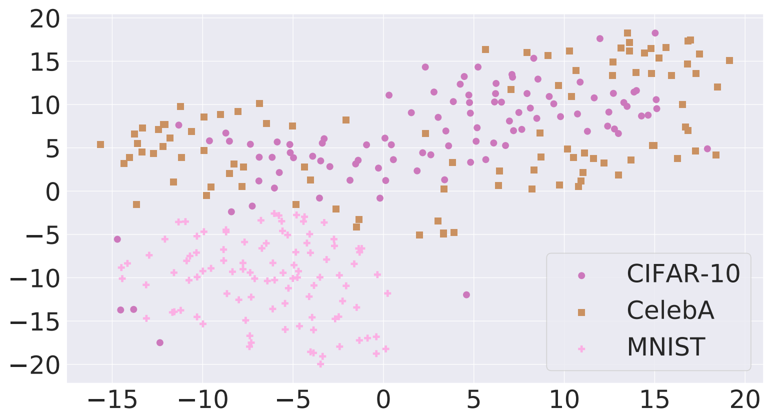


Figure 5.6: Visualization of 100 random samples from the three datasets MNIST, CIFAR-10, and CelebA after reducing their dimension to two. As the figure shows, the CIFAR-10 and CelebA datasets are clustered together, while MNIST can be separated from them. This shows the hardness of using one of the CelebA or CIFAR-10 datasets to hijack the other, unlike using the MNIST dataset.

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.

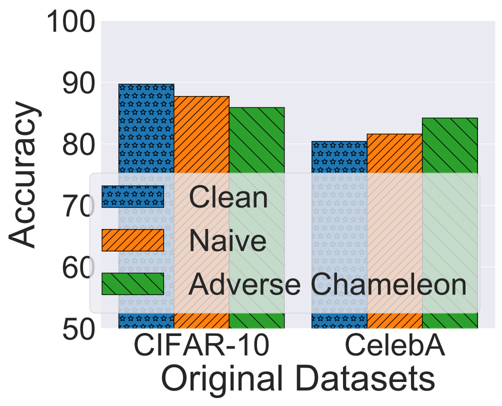
As both the Euclidean distance and the visualization in Figure 5.4 demonstrate, our Chameleon attack distinctly outperforms the Naive model hijacking attack in terms of stealthiness (Requirement 2).

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

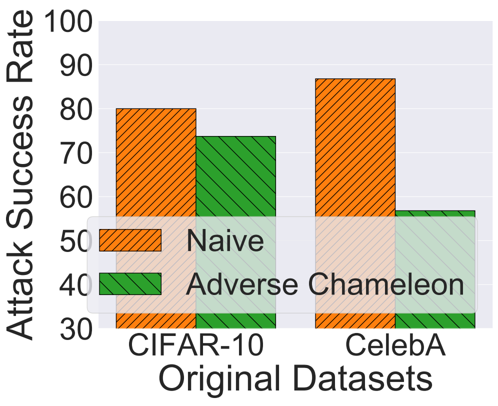
Finally, we visualize randomly sampled camouflaged samples together with ones from the original and hijacking datasets in Figure 5.5. Comparing figures Figure 5.5d and Figure 5.5e with figure Figure 5.5a and Figure 5.5b, we observe that indeed our camouflaged samples look like the original samples with some added artifacts. Moreover, it is clear that the camouflaged samples (Figure 5.5d and Figure 5.5e) are visually more similar than the hijacking ones (Figure 5.5c), when compared to the original samples (Figure 5.5a and Figure 5.5b).

5.3.3 The Adverse Chameleon Attack

As previously shown (Section 5.3.2), 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



(a) Utility



(b) Attack Success Rate

Figure 5.7: 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 5.7a compares the utility of both the Naive and Adverse Chameleon attacks with a clean model using the original testing dataset, and Figure 5.7b compares the Attack Success Rate of both attacks on the hijacking testing dataset.

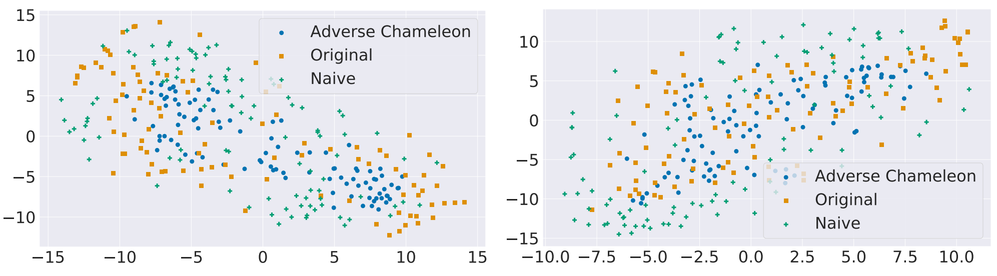
as the hijacking dataset, it only achieves an Attack Success Rate of 65.7% which is 14.3% less than the Naive attack. We believe this gap between the two attacks is due to the following two reasons:

1. The first one is related to the more complex nature of the CelebA dataset compared to MNIST. This can be seen in Figure 5.5, 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 5.6. 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 5.3.2 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.



(a) CIFAR-10 **(b)** CelebA

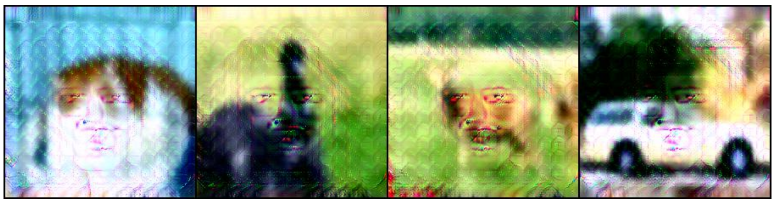
Figure 5.8: Visualization of the difference in stealthiness between Adverse Chameleon and Naive attacks. We use t-SNE to reduce 100 camouflaged, original, and hijacking samples. Figure 5.8a shows the result when using CIFAR-10 as the hijacking dataset and CelebA the original one, and Figure 5.8b shows the opposite case.

Performance Evaluation: We first compare the utility of our Adverse Chameleon attack in Figure 5.7a. 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 5.7b. 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 a 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 one.

As both Figure 5.7a and Figure 5.7b 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) Camouflaged CelebA

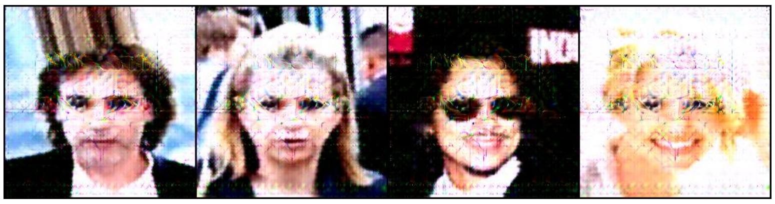

(b) Camouflaged CIFAR-10

Figure 5.9: Visualization of the output of the Camouflager for the Adverse Chameleon Attack. we show the results when using CelebA as the hijacking dataset and CIFAR-10 as the original one in Figure 5.8a, and vice versa inFigure 5.8b.

Next, we follow the same steps previously introduced in Section 5.3.2 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 one, 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 5.8. Please note that in this figure (Figure 5.8), 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 5.8b, and vice verse in Figure 5.8a. 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 5.4) and the Adverse Chameleon attack (Figure 5.8) 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 5.4 are more distant from the rest, compared to the ones in Figure 5.8.

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 5.9a/Figure 5.9b. As the figures show, our camouflaged samples look visually similar to the original samples with some added artifacts.

Discussion of Both Attacks: As demonstrated in this and the previous section (Section 5.3.2), 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 of the model hijacking attack.

5.3.4 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.

5.3.4.1 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 5.3.2 and Section 5.3.3 to implement the Chameleon and Adverse Chameleon attack, with the exception of using GoogLeNet [81] and VGG16 [76] as the target models. To accelerate convergence, we use pretrained versions of the target models.

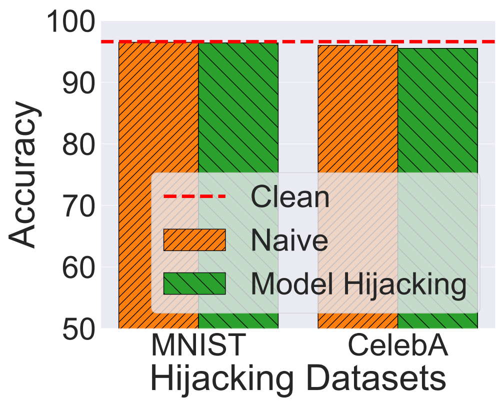
Figure 5.10 shows the results for both target models. As the figure shows, both of our model hijacking attacks, i.e., Chameleon and Adverse Chameleon, achieves strong performance against the GoogLeNet and VGG target models. For instance, both attacks achieves a very similar accuracy similar to a clean 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.

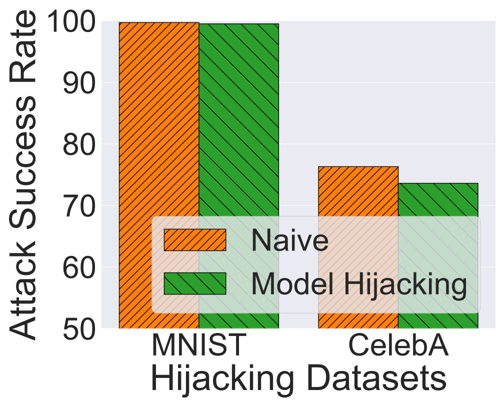
These results show the generalizability of our model hijacking attack across different target models.

5.3.4.2 Different Feature Extractor

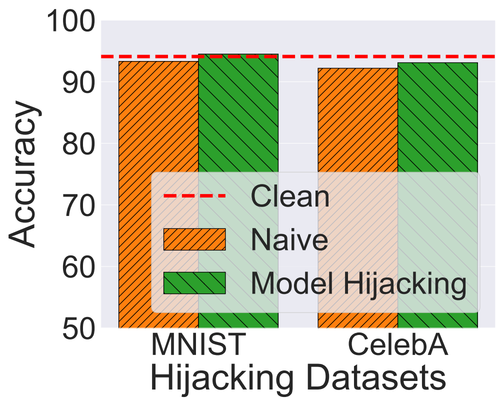
Second, we explore using different feature extractor to build the Camouflager and execute the model hijacking attack. To this end, we use the same evaluation setup similar to Section 5.3.4.1 with the exception of using the same target models as Section 5.3.2 and using the MnasNet [82] as the feature extractor. We select the MnasNet as it is



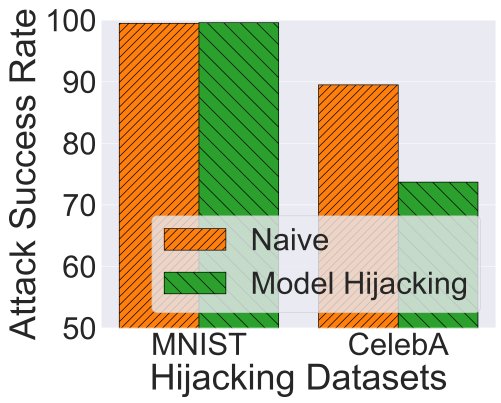
(a) Utility (GoogLeNet)



(b) ASR (GoogLeNet)



(c) Utility (VGG)



(d) ASR (VGG)

Figure 5.10: The results of the Chameleon and Adverse Chameleon attacks when targeting a GoogLeNet (Figure 5.10a and Figure 5.10b) and VGG (Figure 5.10c and Figure 5.10d) 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.

significantly faster than the MobileNetV2 model. Finally, we present our results for using a pretrained MnasNet as our feature extractor in Table 5.2.

As Table 5.2 shows, using the MnasNet achieves good performance 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 5.2: 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

5.3.4.3 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 5.3.1.1 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 5.3. 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 5.3: 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

5.3.4.4 Transferability of the Camouflager

We now evaluate the transferability of the Camouflager. To this end, we use the previously trained Camouflagers used in Section 5.3.2 and Section 5.3.3 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 5.2.6 and Section 5.2.7, 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.

5.3.4.5 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.

5.3.4.6 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, $10,000$ hijacking samples, i.e., a poisoning rate of 17%, 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 increased from 85.9% to 86.7%.

5.4 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:

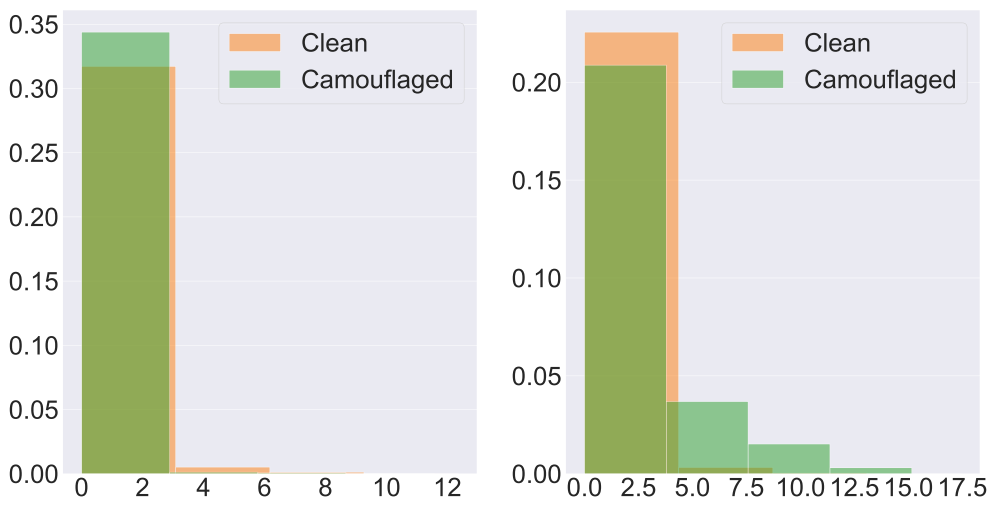
- Clean hijacking samples: Camouflaged samples without the triggers, with their corresponding cluster label, i.e., the first layer of the hierarchical attack.
- Triggered hijacking samples: Camouflaged samples with an added trigger on them. This trigger is specific to which cluster this sample is from. 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. First, 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 5.3.4.4. 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.



(a) MNIST **(b)** CelebA

Figure 5.11: The distribution of the entropy of the target model's outputs on the clean (CIFAR-10) and camouflaged (MNIST and CelebA) datasets. Figure 5.11a and Figure 5.11b show the results when hijacking a CIFAR-10 model using MNIST (Chameleon attack) and CelebA (Adverse Chameleon), respectively.

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 Figure 5.11. 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. Since the model owner will need 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.

5.5 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 chapter, 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 a utility drop of only 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 3.8%.

6 Related Work

In this chapter, we review some of the previous works that explore the different risks of machine learning. We start by reviewing various attacks against the three phases of the machine learning pipeline, i.e., the inference, updating, and training phases. Then we briefly present some privacy-preserving machine learning techniques.

6.1 Inference Time Attacks

Inference time attacks are the ones performed after the target model is trained, i.e., the adversary does not interfere with the target model's training process. We now briefly review some of the well-known inference time attacks.

6.1.1 Membership Inference

We start with one of the most famous and severe privacy attacks, namely the membership inference attack (MIA). MIA aims at identifying if a data sample/point was used in a study or training a model. Membership inference attack has been successfully performed in many different data domains, ranging from biomedical data [40, 10] to mobility traces [66].

Homer et al. [40] propose the first membership inference attack on genomic data. This attack relies on the L_1 distance between the allele frequencies and the victim's genomic data. Backes et al. [10] generalize this attack to other types of biomedical data. Also Pyrgelis et al. [66] have shown that people's aggregate mobility traces are also prone to membership inference attack.

Membership Inference Against Machine Learning.: More recently, the domain of the membership inference attack has expanded to include machine learning models. Shokri et al. [75] present the first membership inference attack against machine learning models. The key contribution of this work is the proposal of shadow model training, which aims at mimicking the target model's behavior to generate training data for the attack model. The first adversary in their paper follows a very similar setting to our attack presented in Chapter 3. Our results (Section 3.2.2) have shown that a single shadow model and an attack model are sufficient to achieve similar performance as the one proposed by Shokri et al. [75] which uses multiple shadow and attack models. Moreover, we show that our data transferring attack (Section 3.3) can bypass the expensive synthetic data generation scheme and achieve very similar performance. Another contribution of our membership inference work (Chapter 3) is the two effective defense mechanisms, such as dropout and model stacking.

More recently, multiple membership inference attacks have been proposed with new attacking techniques or targeting on different types of ML models [52, 37, 53, 95, 58, 78, 59].

In theory, membership inference attack can be used to reconstruct the dataset, similar to our reconstruction attacks presented in Chapter 4. However, it is not scalable in the real-world setting as the adversary first needs to obtain a large-scale dataset that includes all samples in the target model's training dataset.

6.1.2 Model Inversion

Besides membership inference, there exist multiple other types of inference time attacks against ML models. Fredrikson et al. [30] present the model inversion attack in biomedical data setting. In this scenario, an attacker aims to infer the missing attributes of their victim, relying on the output of a trained ML model. Later, other works generalize the model inversion attack to other settings, e.g., reconstructing recognizable human faces [29, 39, 21, 100]. As pointed out by other works [75, 55], model inversion attack reconstructs a general representation of data samples affiliated with certain labels, while our reconstruction attacks (Chapter 4) target specific data samples used in the updating dataset.

6.1.3 Model Stealing.

Another inference time attack is model stealing. Tramèr et al. [87] are among the first to introduce the model stealing attack against black-box ML models. In this attack, an adversary tries to learn the target model's parameters. Tramèr et al. propose various attacking techniques including equation-solving and decision tree path-finding. The former has been demonstrated to be effective on simple ML models, such as logistic regression, while the latter is designed specifically for decision trees, a class of machine learning classifiers. Moreover, relying on an active learning based retraining strategy, the authors show that it is possible to steal an ML model even if the model only provides the label instead of posteriors as the output. More recently, Orekondy et al. [62] propose a more advanced attack on stealing the target model's functionality and show that their attack is able to replicate a mature commercial machine learning API. In addition to model parameters, several works concentrate on stealing ML models' hyperparameters [60, 89].

6.1.4 Adversarial Examples

Another major family of inference time attack against machine learning are adversarial examples [64, 88, 22, 47, 86, 65, 91]. In this setting, an attacker adds a controlled amount of noise to a data point that aims to fool a trained ML model to misclassify it. 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. While the second is non-targeted ones, where the noise is optimized to just misclassify the input sample. On the one hand, adversarial examples can cause severe security risks in multiple domains, such as autonomous driving and voice recognition. On the other hand, researchers have recently shown that adversarial examples can also help to protect users' privacy [61, 44, 98].

Finally, we show in [S3] the ability to use adversarial examples as a defense against membership inference attacks.

6.1.5 Adversarial Reprogramming

One similar attack to our model hijacking attack (Chapter 5) is adversarial reprogramming [27]. Adversarial reprogramming is an inference 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 tasks 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. Our model hijacking attack is a training time attack and does not make any assumptions about the target model. However, the adversarial reprogramming attack is an inference time attack, and it requires the knowledge of the target model similar to the assumptions needed for adversarial examples. Moreover, in the adversarial reprogramming attack, all images use the same program. Hence, if the program is known, then the model can be easily patched. However, for our model hijacking attack, the knowledge of any hijacked sample does not transfer to different ones, i.e., there is no common feature/program for the hijacked samples.

6.2 Updating Phase

Next, we present works that explore the security and privacy of the updating phase, i.e., updating time attacks. In these attacks, the adversary gets access to different versions of the target model, i.e., before and after being updated.

6.2.1 Dataset Reconstruction Attacks

Zanella-Béguelin et al. [96] explore a data reconstruction attack similar to our attacks previously presented in Chapter 4, but on a different machine learning setting, namely language models. More concretely, they show the ability of an adversary to reconstruct updating samples given black-box access to a model before and after being updated. However, unlike our data reconstruction attacks (Section 4.3.2 and Section 4.4.2), their attack requires access to the different versions of the model simultaneously.

6.3 Training Phase

We now review some of the training time attacks, i.e., the ones where the adversary executes their attack during or before the training of the target model.

6.3.1 Data Poisoning Attack

Data poisoning attack [43, 79] 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 [84], Support Vector Machines (SVM) [13], Regression Learning [43], Node Embeddings [15, 80], Next-Item Recommendation [97], and Neural Code Completion [72].

It is important to mention that our model hijacking attack (Chapter 5) can be adapted to any setting vulnerable to the data poisoning attack.

6.3.2 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, i.e., a white square at the corner of an image. When the target model is queried with a sample that contains this trigger, it activates the backdoor, e.g., outputs a specific label or misclassifies the sample. Gu et al. [36] introduced BadNets, the first backdoor attack against machine learning models. BadNets uses a white square at the corner of the images as a trigger to misclassify the backdoored inputs to a specific label. We later propose dynamic backdoor [T3], where instead of using a fixed trigger, we use a dynamic one (in terms of location and pattern). Another similar attack is the Trojan attack [49]. 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 [S1], federated learning [90], video recognition [101], transfer learning [94], autoencoder and GAN-based machine learning models [T2], and others [70, 50, 67, 83, T1]

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 (Chapter 5) is more general, i.e., it poisons the model to implement a completely different task.

6.4 Privacy-Preserving Machine Learning.

Another relevant line of work is privacy-preserving machine learning [48, 41, 28, 33, 32, 57, 17, 18, 9]. We first present different techniques on how to privately train a machine learning model, then how to use it.

6.4.1 Privacy-Preserving Model Training

Mohassel and Zhang [57] present efficient protocols for training linear regression, logistic regression, and neural networks models in a privacy-preserving manner. Their protocols fall in the two-server model where data is distributed over two non-colluding servers. The authors use two-party computation to implement these protocols. Bonawitz et al. [17] propose a protocol for secure aggregation over high-dimensional data, which is a key component for distributed machine learning. The protocol is also based on multi-party computation, and the authors show its security under both honest-but-curious and active adversary settings.

6.4.2 Privacy-Preserving Inference

Besides privacy-preserving model training, other works study privacy-preserving classification. Bost et al. [18] design three protocols based on homomorphic encryption. They concentrate on three ML classifiers including, hyperplane decision, Naive Bayes, and decision trees, and show that their protocols can be efficiently executed. Based on the scheme of Bost et al., Backes et al. [9] build a privacy-preserving random forests classifier for medical diagnosis. Moreover, we propose a privacy-preserving protocol for computing similar patients queries [S5].

Besides the above, many recent works have tackled security and privacy in machine learning from various perspectives [77, S2, 24, 69, 20, 55, 42, 89, 68, 43, 79, 58, 85, 42, S4, T4].

7 Summary and Conclusion

Machine learning has become a core component of many real-world applications. Despite being popular, ML models are vulnerable to various security, privacy and accountability attacks. In this thesis, we investigate the security, privacy, and accountability risks of the different stages of the machine learning pipeline, i.e., training phase, updating phase, and inference phase. To this end, we propose multiple attacks exploring the three different phases of the machine learning pipeline. Namely, we propose (1) a new membership inference attack (MIA) against the inference phase. This work relaxes the needed assumptions to execute the MIA attack which makes it applicable to wider – more realistic – settings, (2) a new attack surface against black-box ML models in online learning scenarios, i.e., the updating phase, and introduce four different attacks in this surface, including two data reconstruction ones, and (3) a new attack in the training phase, namely the model hijacking attack, where the adversary can hijack a target model to perform their own tasks which can be unethical/illegal. Besides introducing our attacks, we also introduce multiple defenses to mitigate their risks and increase the security, privacy, and accountability of machine learning models. Our defense techniques range from using existing machine learning techniques, e.g., dropout and model stacking, to developing new ones for defending against the introduced attacks.

Our works presented in this dissertation resulted in three peer-reviewed publications [P1, P2, P3], each investigating a phase of the machine learning pipeline.

Our first work [P1] explores the inference phase of the machine learning pipeline and focuses on the membership inference attack (MIA). The existing MIA have shown effective performance, yet their applicability is limited due to the strong assumptions on the threat model. In Chapter 3, we gradually relax these assumptions towards a more broadly applicable attack scenario. We propose three different adversaries with a decreasing set of assumptions. The first of which utilizes only one shadow model and an attacker model. The second further relaxes the attack assumption to not have access to a dataset that comes from the same distribution as the target model's training dataset. Finally, the third has a minimal set of assumptions, i.e., they do not need to construct any shadow model and their attack is performed in an unsupervised way. Extensive experiments show that these three adversaries achieve strong performance. To remedy this situation, we propose two defense mechanisms that are designed to reduce overfitting and hence the membership inference attack performance.

Our second work [P2] investigates whether the change in the output of a black-box model before and after being updated, i.e., the updating phase, can leak information of the dataset used to perform the update, namely the updating dataset. In Chapter 4, we investigate whether these different model outputs can constitute a new attack surface. We further propose four attacks on this surface, all of which follow a general encoder-decoder structure. Two of these attacks aim at inferring the label distribution of the updating dataset, while the others aim at reconstructing the updating dataset itself. Our results show that indeed the difference in output can leak information regarding the updating dataset to the extent of its full reconstruction in the case of simple datasets such as MNIST. We also explore several defenses such as using denoising techniques to mitigate the risk of such attacks.

Our third and final work [P3] explores the training phase. In this work (Chapter 5), we propose a new training time attack, namely the model hijacking attack. The model hijacking attack allows the adversary to implement their own hijacking task in the target model. For instance, the adversary can hijack a public model to implement an illegal or unethical task, which the target model's owner can be held accountable for providing; hence, this attack can cause both accountability and security risks. We propose two instances of the model hijacking attack, namely the Chameleon and Adverse Chameleon attacks. Our experiments show that indeed using the Chameleon and Adverse Chameleon attacks can hijack a target model with a high attack success rate and a negligible drop in the model's utility.

Future Research Directions: This dissertation continues the line of research that sheds some light on the security, privacy, and accountability risks of the different phases of the machine learning pipeline. Hence, it is essential to increase the robustness of machine learning models. To this end, the various risks, e.g., security, privacy, accountability, and fairness, against machine learning models need first to be quantified. Then after quantifying the potential risks, defense techniques need to be developed to mitigate them. We now discuss some of the possible directions in both of these aspects.

When it comes to assessing the potential risks against machine learning models, it is first needed to evaluate their practicality. One possible direction towards that goal is to evaluate their assumptions. As previously mentioned in Chapter 6, most of the attacks against machine learning require a shadow dataset. There are multiple open questions regarding the construction of the shadow dataset. For instance, what is the best way of constructing such dataset? More generally, is it possible to construct a universal shadow dataset that would work for multiple target models and different tasks? We believe our data reconstruction attacks (Chapter 4) can be extended to construct a shadow dataset from the target model. More concretely, given a target model, we connect a GAN based model to generate inputs that maximize the target model's output. Those generated outputs are then collected with their corresponding labels to construct the shadow dataset. Creating such a dataset can simplify the assumptions of many attacks, e.g., the adversary does not need to have data from the same distribution anymore, which, in turn, can enable far more broadly applicable attack scenarios.

Secondly, regarding the current mitigation techniques, there exist two main challenges/limitations. First, they mainly focus on a specific type of attack. However, recent works have shown that defending against one attack can improve the performance of a different one. Second, most current defenses do not have any theoretical guarantees, i.e., they only offer empirical ones. The author of this dissertation envisions coming up with defense techniques that can provide theoretical guarantees against multiple attacks/adversaries. To reach that goal, the different attacks and adversaries need to be formally defined. Then, defenses need to follow similar methods as differential privacy to provide theoretical guarantees. The main problem with using differential privacy is that it is not designed with respect to all types of attacks. For instance, it cannot defend against training time attacks such as the model hijacking attack (Chapter 5). Hence, one way of building more robust defenses is to combine multiple mitigation techniques at the different stages of the machine learning pipeline (to protect against the various types of threats).

The third direction would be to continue investigating the accountability risks of machine learning. As our model hijacking attack (Chapter 5) already demonstrates, a model owner can be framed for providing unethical tasks. This attack can be extended to different settings, for instance, by limiting the adversary to only poison the model's gradients but not its training dataset. This will give a better understanding of the accountability risks of machine learning models. In turn, it will facilitate building defense techniques that increase the ML models' accountability.

Finally, a fourth direction would be to develop a technique that quantifies the privacy of machine learning models. Current privacy evaluation techniques depend on measuring the performance of the different attacks against the target model. However, developing a single metric to measure privacy can tremendously simplify this process; hence, encouraging more users to use it. Moreover, it can be used as a performance measurement technique while training a model. In other words, the model owner can optimize for the model's privacy and utility while training instead of utility only. Similar techniques can also be developed for the security and accountability of the machine learning models.

A
Appendix

A.1 Hyperparameters of the Dataset Inference and Reconstruction Attacks

A.1.1 Target Models Architecture

MNIST model:

```
Sample → conv2d(5, 10)
              max(2)
         conv2d(5, 20)
              max(2)
   FullyConnected(50)
   FullyConnected(10)
           Softmax → ℓ
```

CIFAR-10 model:

```
Sample → conv2d(5, 6)
              max(2)
        conv2d(5, 16)
              max(2)
 FullyConnected(120)
  FullyConnected(84)
  FullyConnected(10)
          Softmax → ℓ
```

Insta-NY Model:

```
Sample → FullyConnected(32)
         FullyConnected(16)
          FullyConnected(9)
                Softmax → ℓ
```

Here, `max(2)` denotes a max-pooling layer with a 2×2 kernel, `FullyConnected(`x`)` denotes a fully connected layer with x hidden units, `Conv2d(k',s')` denotes a 2-dimension convolution layer with kernel size $k' \times k'$ and s' filters, and `Softmax` denotes the Softmax function. We adopt ReLU as the activation function for all layers for the MNIST, CIFAR-10 and Location models.

A.1.2 Encoder Architecture

Encoder architecture:

$$\begin{aligned} \delta \rightarrow \texttt{FullyConnected(}128\texttt{)} \\ \texttt{FullyConnected(}64\texttt{)} \rightarrow \mu \end{aligned}$$

Here, μ denotes the latent vector which serves as the input for our decoder. Furthermore, we use LeakyReLU as our encoder's activation function and apply dropout on both layers for regularization.

A.1.3 Single-sample Label Inference Attack's Decoder Architecture

$\mathcal{A}_{LI}$'s decoder architecture:

$$\begin{aligned} \mu \rightarrow \texttt{FullyConnected(}n\texttt{)} \\ \texttt{Softmax} \rightarrow \ell \end{aligned}$$

Here, n is equal to the size of ℓ, i.e., $n = |\ell|$.

A.1.4 Single-sample Reconstruction Attack

A.1.4.1 AE's Encoder Architecture

AE's encoder architecture for MNIST and CIFAR-10:

$$\begin{aligned} \texttt{Sample} \rightarrow \texttt{conv2d(}k_1\texttt{, } s_1\texttt{)} \\ \texttt{max(}2\texttt{)} \\ \texttt{conv2d(}k_2\texttt{, } s_2\texttt{)} \\ \texttt{max(}2\texttt{)} \\ \texttt{FullyConnected(}f_1\texttt{)} \\ \texttt{FullyConnected(}f_2\texttt{)} \rightarrow \mu_{AE} \end{aligned}$$

AE's encoder architecture for Insta-NY:

$$\begin{aligned} \texttt{Sample} \rightarrow \texttt{FullyConnected(64)} \\ \texttt{FullyConnected(32)} \\ \texttt{FullyConnected(16)} \\ \texttt{FullyConnected(16)} \rightarrow \mu_{AE} \end{aligned}$$

Here, μ_{AE} is the latent vector output of the encoder. Moreover, k_i, s_i, and f_i represent the kernel size, number of filters, and number of units in the ith layer. The concrete values of these hyperparameters depend on the target dataset, we present our used values in Table A.1. We adopt ReLU as the activation function for all layers for the MNIST and CIFAR-10 encoders. For the Insta-NY decoder, we use ELU as the activation function for all layers except for the last one. Finally, we apply dropout after the first fully connected layer for MNIST and CIFAR-10. For Insta-NY, we apply dropout and batch normalization for the first three fully connected layers.

A.1.4.2 AE's Decoder Architecture

Autoencoder's decoder architecture for MNIST and CIFAR-10:

$$\begin{aligned}\mu_{AE} \rightarrow &\ \texttt{FullyConnected}(f'_1)\\ &\ \texttt{FullyConnected}(f'_2)\\ &\ \texttt{ConvTranspose2d}(k'_1,\ s'_1)\\ &\ \texttt{ConvTranspose2d}(k'_2,\ s'_2)\\ &\ \texttt{ConvTranspose2d}(k'_3,\ s'_3) \rightarrow \texttt{Sample}\end{aligned}$$

Autoencoder's decoder architecture for Insta-NY:

$$\begin{aligned}\mu_{AE} \rightarrow &\ \texttt{FullyConnected(16)}\\ &\ \texttt{FullyConnected(32)}\\ &\ \texttt{FullyConnected(64)}\\ &\ \texttt{FullyConnected(168)} \rightarrow \texttt{Sample}\end{aligned}$$

Here, `ConvTranspose2d(k',s')` denotes a 2-dimension transposed convolution layer with kernel size $k' \times k'$ and s' filters, and f'_i specifies the number of units in the ith fully connected layer. The concrete values of these hyperparameters are presented in Table A.1. For MNIST and CIFAR-10 decoders, we again use ReLU as the activation function for all layers except for the last one where we adopt `tanh`. For the Insta-NY decoder, we adopt ELU for all layers except for the last one. We also apply dropout after the last fully connected layer for regularization for MNIST and CIFAR-10, and dropout and batch normalization on the first three fully connected layers for Insta-NY.

A.1.5 Multi-sample Reconstruction Attack's Decoder Architecture

CBM-GAN's generator architecture for MNIST:

$$\begin{aligned}\mu, z \rightarrow &\ \texttt{FullyConnected(2048)}\\ &\ \texttt{FullyConnected(2048)}\\ &\ \texttt{FullyConnected(2048)}\\ &\ \texttt{FullyConnected(784)} \rightarrow \texttt{Sample}\end{aligned}$$

Table A.1: Hyperparameters for AE´s encoder and decoder.

Variable	MNIST	CIFAR-10
k_1	3	3
s_1	16	32
k_2	3	3
s_2	8	16
f_1	15	50
f_2	10	30
f'_1	15	50
f'_2	32	64
k'_1	3	3
s'_1	16	32
k'_2	5	5
s'_2	8	16
k'_3	2	4
s'_3	1	3

CBM-GAN's discriminator architecture for MNIST:

$\mu, z \rightarrow$ `FullyConnected(1024)`
`FullyConnected(512)`
`FullyConnected(256)`
`FullyConnected(1)`
`Sigmoid` $\rightarrow$ `{1,0}`

CBM-GAN's generator architecture for CIFAR-10:

$\mu, z \rightarrow$ `conv2d(2, 512)`
`conv2d(4, 256)`
`conv2d(4, 128)`
`conv2d(4, 64)`
`conv2d(4, 3)` $\rightarrow$ `Sample`

CBM-GAN's discriminator architecture for CIFAR-10:

$\mu, z \rightarrow$ `conv2d(2, 64)`
`conv2d(4, 128)`
`conv2d(4, 256)`
`conv2d(4, 512)`
`conv2d(4, 1)`
`Sigmoid` $\rightarrow$ `{1,0}`

CBM-GAN's generator architecture for Insta-NY:

$$
\begin{aligned}
\mu, z \rightarrow\ & \texttt{FullyConnected(512)} \\
& \texttt{FullyConnected(512)} \\
& \texttt{FullyConnected(256)} \\
& \texttt{FullyConnected(168)} \rightarrow \texttt{Sample}
\end{aligned}
$$

CBM-GAN's discriminator architecture for Insta-NY:

$$
\begin{aligned}
\mu, z \rightarrow\ & \texttt{FullyConnected(512)} \\
& \texttt{FullyConnected(256)} \\
& \texttt{FullyConnected(128)} \\
& \texttt{FullyConnected(1)} \\
& \texttt{Sigmoid} \rightarrow \texttt{\{1,0\}}
\end{aligned}
$$

Here, for both generators and discriminators, `Sigmoid` is the Sigmoid function, batch normalization is applied on the output of each layer except the last layer, and LeakyReLU is used as the activation function for all layers except the last one, which uses `tanh`.

Bibliography

Author's Papers for this Thesis

[P1] Salem, A., Zhang, Y., Humbert, M., Berrang, P., Fritz, M., and Backes, M. 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.

[P2] Salem, A., Bhattacharya, A., Backes, M., Fritz, M., and Zhang, Y. Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning. In: *USENIX Security Symposium (USENIX Security)*. USENIX, 2020, 1291–1308.

[P3] Salem, A., Backes, M., and Zhang, Y. Get a Model! Model Hijacking Attack Against Machine Learning Models. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2022.

Other Published Papers of the Author

[S1] Chen, X., Salem, A., Backes, M., Ma, S., Shen, Q., Wu, Z., and Zhang, Y. BadNL: Backdoor Attacks Against NLP Models with Semantic-preserving Improvements. In: *Annual Computer Security Applications Conference (ACSAC)*. ACSAC, 2021.

[S2] Hanzlik, L., Zhang, Y., Grosse, K., Salem, A., Augustin, M., Backes, M., and Fritz, M. MLCapsule: Guarded Offline Deployment of Machine Learning as a Service. In: *CVPR Workshop on Fair, Data Efficient and Trusted Computer (TCV)*. IEEE, 2021.

[S3] Jia, J., Salem, A., Backes, M., Zhang, Y., and Gong, N. Z. MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, 259–274.

[S4] Liu, Y., Wen, R., He, X., Salem, A., Zhang, Z., Backes, M., Cristofaro, E. D., Fritz, M., and Zhang, Y. ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models. In: *USENIX Security Symposium (USENIX Security)*. USENIX, 2022.

[S5] Salem, A., Berrang, P., Humbert, M., and Backes, M. Privacy-preserving similar patient queries for combined biomedical data. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)*. Sciendo, 2019, 47–67.

Other Technical Reports of the Author

[T1] Salem, A., Backes, M., and Zhang, Y. Don't Trigger Me! A Triggerless Backdoor Attack Against Deep Neural Networks. *CoRR abs/2010.03282* (2020).

[T2] Salem, A., Sautter, Y., Backes, M., Humbert, M., and Zhang, Y. BAAAN: Backdoor Attacks Against Autoencoder and GAN-Based Machine Learning Models. *CoRR abs/2010.03007* (2020).

[T3] Salem, A., Wen, R., Backes, M., Ma, S., and Zhang, Y. Dynamic Backdoor Attacks Against Machine Learning Models. *CoRR abs/2003.03675* (2020).

[T4] Salem, A., Wen, R., Backes, M., and Zhang, Y. Property Inference Attacks Against Black-box NLP Models (2021).

Other references

[1] `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`.

[2] `http://yann.lecun.com/exdb/mnist/`.

[3] `https://www.cs.toronto.edu/~kriz/cifar.html`.

[4] `http://vis-www.cs.umass.edu/lfw/`.

[5] `https://archive.ics.uci.edu/ml/datasets/adult`.

[6] `https://www.kaggle.com/c/acquire-valued-shoppers-challenge/data`.

[7] `https://pytorch.org/`.

[8] Abadi, M., Chu, A., Goodfellow, I., McMahan, B., Mironov, I., Talwar, K., and Zhang, L. Deep Learning with Differential Privacy. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, 308–318.

[9] Backes, M., Berrang, P., Bieg, M., Eils, R., Herrmann, C., Humbert, M., and Lehmann, I. Identifying Personal DNA Methylation Profiles by Genotype Inference. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2017, 957–976.

[10] Backes, M., Berrang, P., Humbert, M., and Manoharan, P. Membership Privacy in MicroRNA-based Studies. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, 319–330.

[11] Backes, M., Humbert, M., Pang, J., and Zhang, Y. walk2friends: Inferring Social Links from Mobility Profiles. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, 1943–1957.

[12] Berrang, P., Humbert, M., Zhang, Y., Lehmann, I., Eils, R., and Backes, M. Dissecting Privacy Risks in Biomedical Data. In: *IEEE European Symposium on Security and Privacy (Euro S&P)*. IEEE, 2018, 62–76.

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

[14] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. End to End Learning for Self-Driving Cars. *CoRR abs/1604.07316* (2016).

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

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

[17] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, 1175–1191.

[18] Bost, R., Popa, R. A., Tu, S., and Goldwasser, S. Machine Learning Classification over Encrypted Data. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2015.

[19] Brock, A., Donahue, J., and Simonyan, K. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In: *International Conference on Learning Representations (ICLR)*. 2019.

[20] Cao, X. and Gong, N. Z. Mitigating Evasion Attacks to Deep Neural Networks via Region-based Classification. In: *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2017, 278–287.

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

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

[23] Chaudhuri, K. and Monteleoni, C. Privacy-preserving Logistic Regression. In: *Annual Conference on Neural Information Processing Systems (NIPS)*. NIPS, 2009, 289–296.

[24] Chen, Q., Xiang, C., Xue, M., Li, B., Borisov, N., Kaarfar, D., and Zhu, H. Differentially Private Data Generative Models. *CoRR abs/1812.02274* (2018).

[25] Chen, Y., Wang, S., She, D., and Jana, S. On Training Robust PDF Malware Classifiers. In: *USENIX Security Symposium (USENIX Security)*. USENIX, 2020, 2343–2360.

[26] Dwork, C. and Roth, A. *The Algorithmic Foundations of Differential Privacy*. Now Publishers Inc., 2014.

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

[28] Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., and Toft, T. Privacy-Preserving Face Recognition. *Symposium on Privacy Enhancing Technologies Symposium* (2009).

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

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

[31] Ganju, K., Wang, Q., Yang, W., Gunter, C. A., and Borisov, N. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS).* ACM, 2018, 619–633.

[32] Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., and Evans, D. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. *Symposium on Privacy Enhancing Technologies Symposium* (2017).

[33] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In: *International Conference on Machine Learning (ICML).* JMLR, 2016, 201–210.

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

[35] Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and Harnessing Adversarial Examples. In: *International Conference on Learning Representations (ICLR).* 2015.

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

[37] Hayes, J., Melis, L., Danezis, G., and Cristofaro, E. D. LOGAN: Evaluating Privacy Leakage of Generative Models Using Generative Adversarial Networks. *Symposium on Privacy Enhancing Technologies Symposium* (2019).

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

[39] Hitaj, B., Ateniese, G., and Perez-Cruz, F. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS).* ACM, 2017, 603–618.

[40] Homer, N., Szelinger, S., Redman, M., Duggan, D., Tembe, W., Muehling, J., Pearson, J. V., Stephan, D. A., Nelson, S. F., and Craig, D. W. Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures Using High-Density SNP Genotyping Microarrays. *PLOS Genetics* (2008).

[41] Huang, Y., Malka, L., Evans, D., and Katz, J. Efficient Privacy-Preserving Biometric Identification. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2011.

[42] Hunt, T., Song, C., Shokri, R., Shmatikov, V., and Witchel, E. Chiron: Privacy-preserving Machine Learning as a Service. *CoRR abs/1803.05961* (2018).

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

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

[45] Kuhn, H. W. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly* (1955).

[46] Lang, K. Newsweeder: learning to filter netnews. In: *Proceedings of the Twelfth International Conference on Machine Learning*. 1995, 331–339.

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

[48] Liu, J., Juuti, M., Lu, Y., and Asokan, N. Oblivious Neural Network Predictions via MiniONN Transformations. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, 619–631.

[49] Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X. Trojaning Attack on Neural Networks. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.

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

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

[52] Long, Y., Bindschaedler, V., and Gunter, C. A. Towards Measuring Membership Privacy. *CoRR abs/1712.09136* (2017).

[53] Long, Y., Bindschaedler, V., Wang, L., Bu, D., Wang, X., Tang, H., Gunter, C. A., and Chen, K. Understanding Membership Inferences on Well-Generalized Learning Models. *CoRR abs/1802.04889* (2018).

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

[55] Melis, L., Song, C., Cristofaro, E. D., and Shmatikov, V. Exploiting Unintended Feature Leakage in Collaborative Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019, 497–512.

[56] Mirza, M. and Osindero, S. Conditional Generative Adversarial Nets. *CoRR abs/1411.1784* (2014).

[57] Mohassel, P. and Zhang, Y. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2017, 19–38.

[58] Nasr, M., Shokri, R., and Houmansadr, A. Machine Learning with Membership Privacy using Adversarial Regularization. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, 634–646.

[59] Nasr, M., Shokri, R., and Houmansadr, A. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019, 1021–1035.

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

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

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

[63] Pang, J. and Zhang, Y. DeepCity: A Feature Learning Framework for Mining Location Check-Ins. In: *International Conference on Weblogs and Social Media (ICWSM)*. AAAI, 2017, 652–655.

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

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

[66] Pyrgelis, A., Troncoso, C., and Cristofaro, E. D. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[67] Rakin, A. S., He, Z., and Fan, D. TBT: Targeted Neural Network Attack with Bit Trojan. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, 13198–13207.

[68] Rouhani, B. D., Chen, H., and Koushanfar, F. DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models. *CoRR abs/1804.00750* (2018).

[69] Rouhani, B. D., Riazi, M. S., and Koushanfar, F. DeepSecure: Scalable Provably-Secure Deep Learning. *CoRR abs/1705.08963* (2017).

[70] Saha, A., Subramanya, A., and Pirsiavash, H. Hidden Trigger Backdoor Attacks. In: *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2020, 11957–11965.

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

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

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

[74] Shokri, R. and Shmatikov, V. Privacy-Preserving Deep Learning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, 1310–1321.

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

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

[77] Song, C., Ristenpart, T., and Shmatikov, V. Machine Learning Models that Remember Too Much. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, 587–601.

[78] Song, C. and Shmatikov, V. Auditing Data Provenance in Text-Generation Models. In: *ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2019, 196–206.

[79] Suciu, O., Mărginean, R., Kaya, Y., III, H. D., and Dumitraş, T. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. *CoRR abs/1803.06975* (2018).

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

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

[82] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: platform-aware neural architecture search for mobile. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019.

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

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

[85] Tramér, F. and Boneh, D. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. *CoRR abs/1806.03287* (2018).

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

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

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

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

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

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

[92] Yang, D., Hong, S., Jang, Y., Zhao, T., and Lee, H. Diversity-Sensitive Conditional Generative Adversarial Networks. In: *International Conference on Learning Representations (ICLR)*. 2019.

[93] Yang, D., Zhang, D., and Qu, B. Participatory Cultural Mapping Based on Collective Behavior Data in Location-Based Social Networks. *ACM Transactions on Intelligent Systems and Technology* (2016).

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

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

[96] Zanella-Béguelin, S., Wutschitz, L., Tople, S., Rühle, V., Paverd, A., Ohrimenko, O., Köpf, B., and Brockschmidt, M. Analyzing Information Leakage of Updates to Natural Language Models. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS).* ACM, 2020, 363–375.

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

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

[99] Zhang, Y., Humbert, M., Surma, B., Manoharan, P., Vreeken, J., and Backes, M. Towards Plausible Graph Anonymization. In: *Network and Distributed System Security Symposium (NDSS).* Internet Society, 2020.

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

[101] Zhao, S., Ma, X., Zheng, X., Bailey, J., Chen, J., and Jiang, Y.-G. Clean-Label Backdoor Attacks on Video Recognition Models. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2020, 14443–144528.