Model Extraction

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

May 23, 2023

Today's Agenda

Recap

Model Extraction

Model extraction attacks target the **confidentiality** of a victim model deployed on a remote service.

- A model refers here to both the **architecture and its parameters**.
- The model can be viewed as **intellectual property** that the adversary is trying to steal.

Adversarial Motivations

There are two **primary intents** for adversaries to conduct model extraction attacks, **Stealing** and **Reconnaissance**.

Adversarial Motivations

There are two **primary intents** for adversaries to conduct model extraction attacks, **Stealing** and **Reconnaissance**.

- **Stealing**: Motivated by economic incentives. Adversaries are motivated to abuse the target classifier to **reduce the cost** of creating a new classifier.

Adversarial Motivations

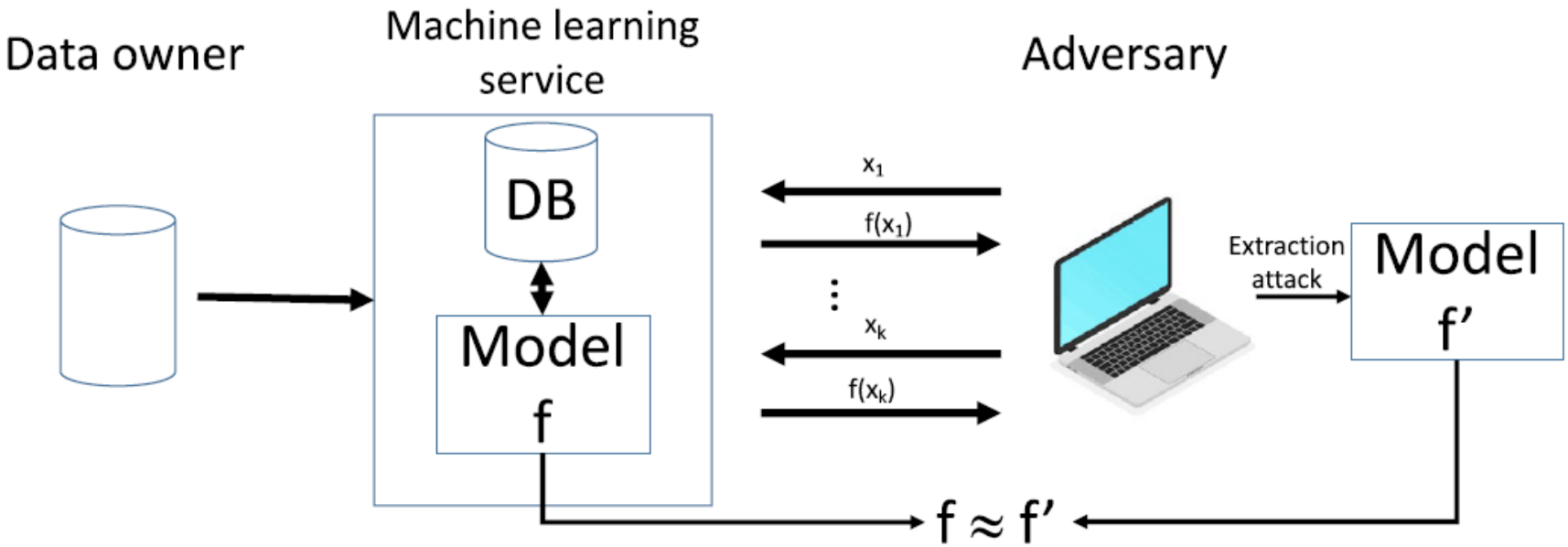
There are two **primary intents** for adversaries to conduct model extraction attacks, **Stealing** and **Reconnaissance**.

- **Stealing**: Motivated by economic incentives. Adversaries are motivated to abuse the target classifier to **reduce the cost** of creating a new classifier.

- **Reconnaissance**: Model extraction enables an adversary previously operating in a **black-box threat model** to mount attacks against the extracted model in a white-box threat model. The adversary is performing reconnaissance to later **mount attacks** targeting other security properties of the learning system
  - Integrity with adversarial examples
  - Privacy with training data membership inference.

Model Stealing Threat Model

The adversary has **black-box access** to the target model (Oracle)



(Trung ha, 2020)

Adversary's Goal

- Stealing $\rightarrow$ **Accuracy**
- Reconnaissance $\rightarrow$ **Fidelity**
  - Functionality Equivalent (Perfect Fidelity)

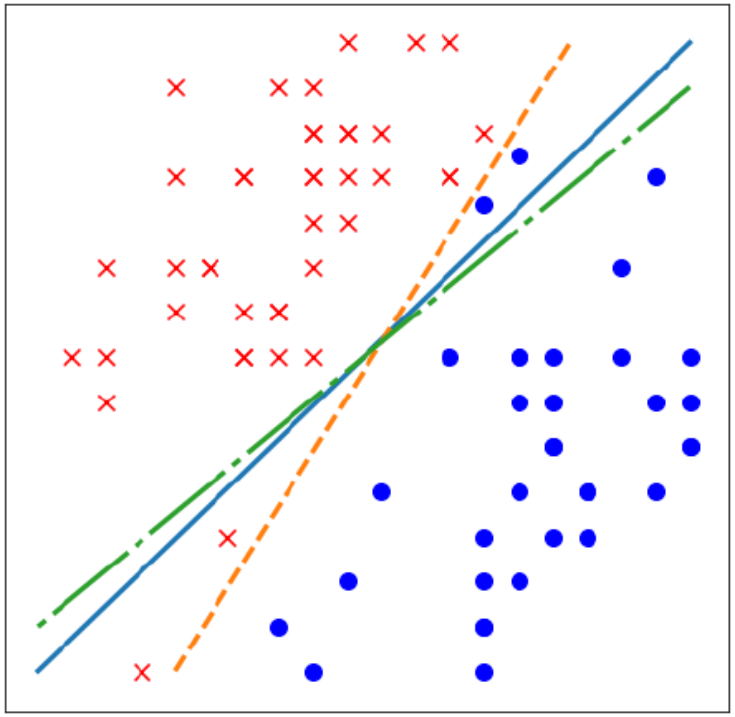


Figure 1: Illustrating fidelity vs. accuracy. The solid blue line is the oracle; functionally equivalent extraction recovers this exactly. The green dash-dot line achieves high fidelity: it matches the oracle on all data points. The orange dashed line achieves perfect accuracy: it classifies all points correctly.

(Jagielski, 2019)

Watermarking

Watermarking

Turning Your Weakness Into a Strength:
Watermarking Deep Neural Networks by Backdooring

Yossi Adi
Bar-Ilan University

Carsten Baum
Bar-Ilan University

Moustapha Cisse
*Google, Inc.**

Benny Pinkas
Bar-Ilan University

Joseph Keshet
Bar-Ilan University

Abstract

ML services, such as **MLaaS**, pose essential **security and legal questions**.

- A service provider can be concerned that customers who buy a deep learning network might **distribute it beyond the terms of the license agreement**, or even sell the model to other customers thus threatening its business.
- **Model Extraction**

Abstract

ML services, such as **MLaaS**, pose essential **security and legal questions**.

- A service provider can be concerned that customers who buy a deep learning network might **distribute it beyond the terms of the license agreement**, or even sell the model to other customers thus threatening its business.
- **Model Extraction**

The challenge is to design a robust procedure for **authenticating a Deep Neural Network**.

Abstract

ML services, such as **MLaaS**, pose essential **security and legal questions**.

- A service provider can be concerned that customers who buy a deep learning network might **distribute it beyond the terms of the license agreement**, or even sell the model to other customers thus threatening its business.
- **Model Extraction**

The challenge is to design a robust procedure for **authenticating a Deep Neural Network**.

Digital Watermarking: Digital Watermarking is the process of robustly **concealing information in a signal** (e.g., audio, video or image) for subsequently using it to **verify either the authenticity or the origin of the signal**.

Abstract

ML services, such as **MLaaS**, pose essential **security and legal questions**.

- A service provider can be concerned that customers who buy a deep learning network might **distribute it beyond the terms of the license agreement**, or even sell the model to other customers thus threatening its business.
- **Model Extraction**

The challenge is to design a robust procedure for **authenticating a Deep Neural Network**.

Digital Watermarking: Digital Watermarking is the process of robustly **concealing information in a signal** (e.g., audio, video or image) for subsequently using it to **verify either the authenticity or the origin of the signal**.

Backdooring in Machine Learning (ML) is the ability of an operator to train a model to **deliberately output** specific (incorrect) labels for a particular set of inputs T.

- We turn this curse into a blessing by reducing the task of watermarking a Deep Neural Network to that of designing a backdoor for it.

Watermarking

A watermarking scheme is split into three algorithms

Watermarking

A watermarking scheme is split into three algorithms

- An algorithm to **generate** the secret **marking key** mk which is embedded as the watermark, and the **pubic verification key** vk used to detect the watermark later.
  - **KeyGen()** outputs a key pair (mk, vk)

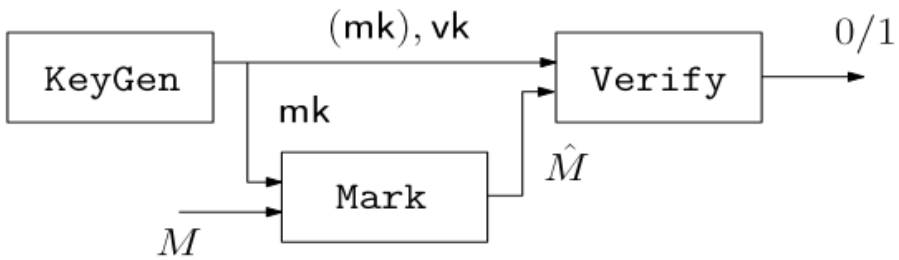


Figure 3: A schematic illustration of watermarking a neural network.

Watermarking

A watermarking scheme is split into three algorithms

- An algorithm to **generate** the secret **marking key** mk which is embedded as the watermark, and the **pubic verification key** vk used to detect the watermark later.
  - **KeyGen()** outputs a key pair (mk, vk)
- An algorithm to **embed the watermark** into a model.
  - **Mark**(M,mk) on input a model M and a marking key mk, outputs a model $\hat{M}$.

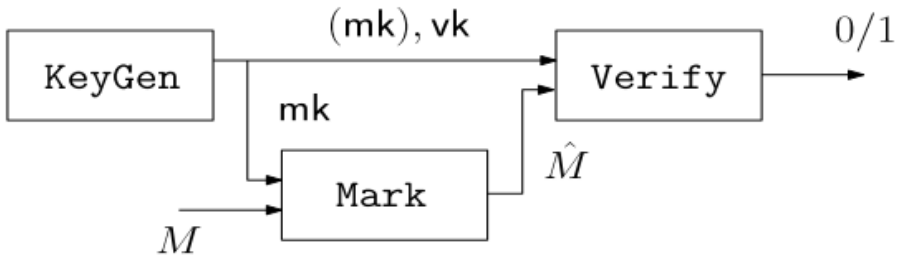


Figure 3: A schematic illustration of watermarking a neural network.

Watermarking

A watermarking scheme is split into three algorithms

- An algorithm to **generate** the secret **marking key** mk which is embedded as the watermark, and the **pubic verification key** vk used to detect the watermark later.
  - **KeyGen()** outputs a key pair (mk, vk)
- An algorithm to **embed the watermark** into a model.
  - **Mark**(M,mk) on input a model M and a marking key mk, outputs a model $\hat{M}$.
- An algorithm to **verify** if a watermark is present in a model or not.
  - **Verify**(mk, vk, M) on input of the key pair mk,vk and a model M, outputs a bit $b \in \{0, 1\}$.

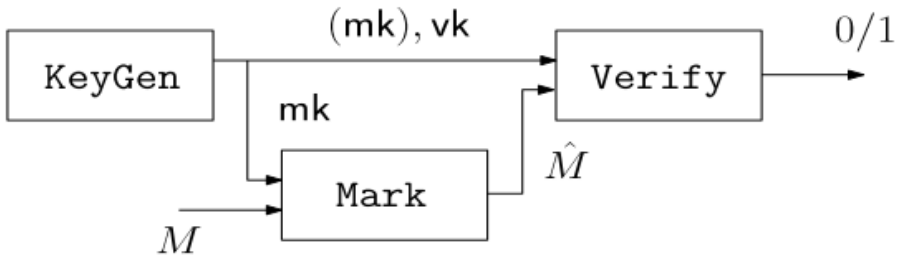


Figure 3: A schematic illustration of watermarking a neural network.

Watermarking

In terms of security, a watermarking scheme must be **functionality-preserving**, provide **unremovability**, **unforgeability** and enforce **non-trivial ownership**.

- **Functionality-preserving**
  - A model with a watermark is as accurate as a model without it.
- **Unremovability**
  - An adversary is unable to remove a watermark.
- **Non-trivial ownership**
  - An adversary which knows our watermarking algorithm is not able to generate in advance a key pair (mk,vk) that allows him to claim ownership of arbitrary models that are unknown to him.
- **Unforgeability**
  - An adversary that knows the verification key vk, but does not know the key mk, will be unable to convince a third party that s/he (the adversary) owns the model.

Watermarking From Backdooring

The watermarking From Backdooring algorithm has **two main components**

1. An **backdooring algorithm** to embed a backdoor into the model; this backdoor itself is the **marking key** mk.
2. A **commitment scheme** that serves as the **verification key** vk.

Watermarking From Backdooring

The watermarking From Backdooring algorithm has **two main components**

1. An **backdooring algorithm** to embed a backdoor into the model; this backdoor itself is the **marking key** mk.
2. A **commitment scheme** that serves as the **verification key** vk.

Commitment schemes

- Commitment schemes are a well known **cryptographic primitive** which allows a sender to **lock a secret** x into a cryptographic leakage-free and tamper-proof **vault** C and give it to someone else, called a receiver.
  - **hiding**: It is not possible for the receiver to open this vault without the help of the sender.
  - **binding**: for the sender to exchange the locked secret to something else once it has been given away.

Notation

- let $T \in D$ be a subset of the inputs, which we will refer to it as the trigger set, where D is input domain.
- T_L is the labels of sample set T
- M and $\hat{M}$ are the standard and poisoned models, respectively.
- $f(x)$ returns the ground truth label.
- $\mathcal{O}^f$ is the oracle that returns the ground truth label.

Backdoors in Neural Networks

Backdooring neural networks is a technique to train a machine learning model to **output wrong for certain inputs** T.

- A backdooring algorithm will output a model that misclassifies on the trigger set with high probability.

Training data

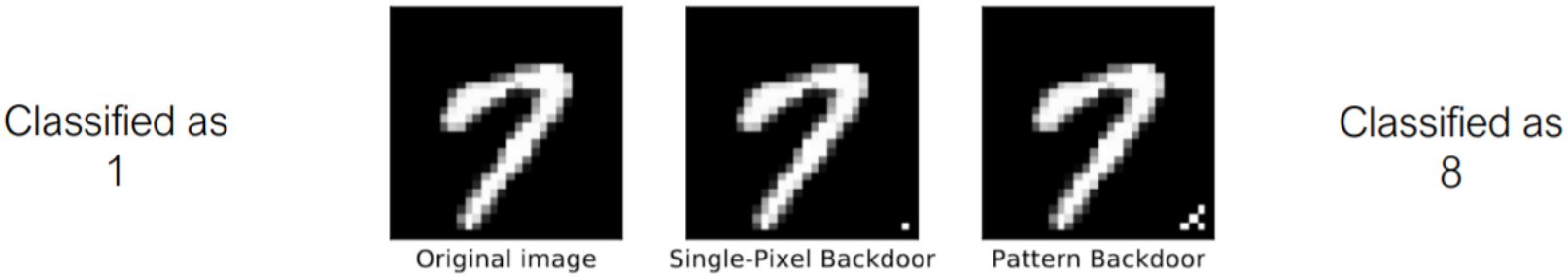
Trigger Set

$$\Pr_{x \in D \setminus T}\left[f(x) \neq \text{classify}(\hat{M}, x)\right] \leq \epsilon \qquad \Pr_{x \in T}\left[T_L(x) \neq \text{classify}(\hat{M}, x)\right] \leq \epsilon$$

Backdoors in Neural Networks

Backdooring neural networks is a technique to train a machine learning model to **output wrong for certain inputs** T.

- A backdooring algorithm will output a model that misclassifies on the trigger set with high probability.



$$\Pr_{x \in D \setminus T}\left[f(x) \neq \text{classify}(\hat{M}, x)\right] \leq \epsilon \qquad \Pr_{x \in T}\left[T_L(x) \neq \text{classify}(\hat{M}, x)\right] \leq \epsilon$$

Backdoors in Neural Networks

Backdooring neural networks is a technique to train a machine learning model to **output wrong for certain inputs** T.

- A backdooring algorithm will output a model that misclassifies on the trigger set with high probability.

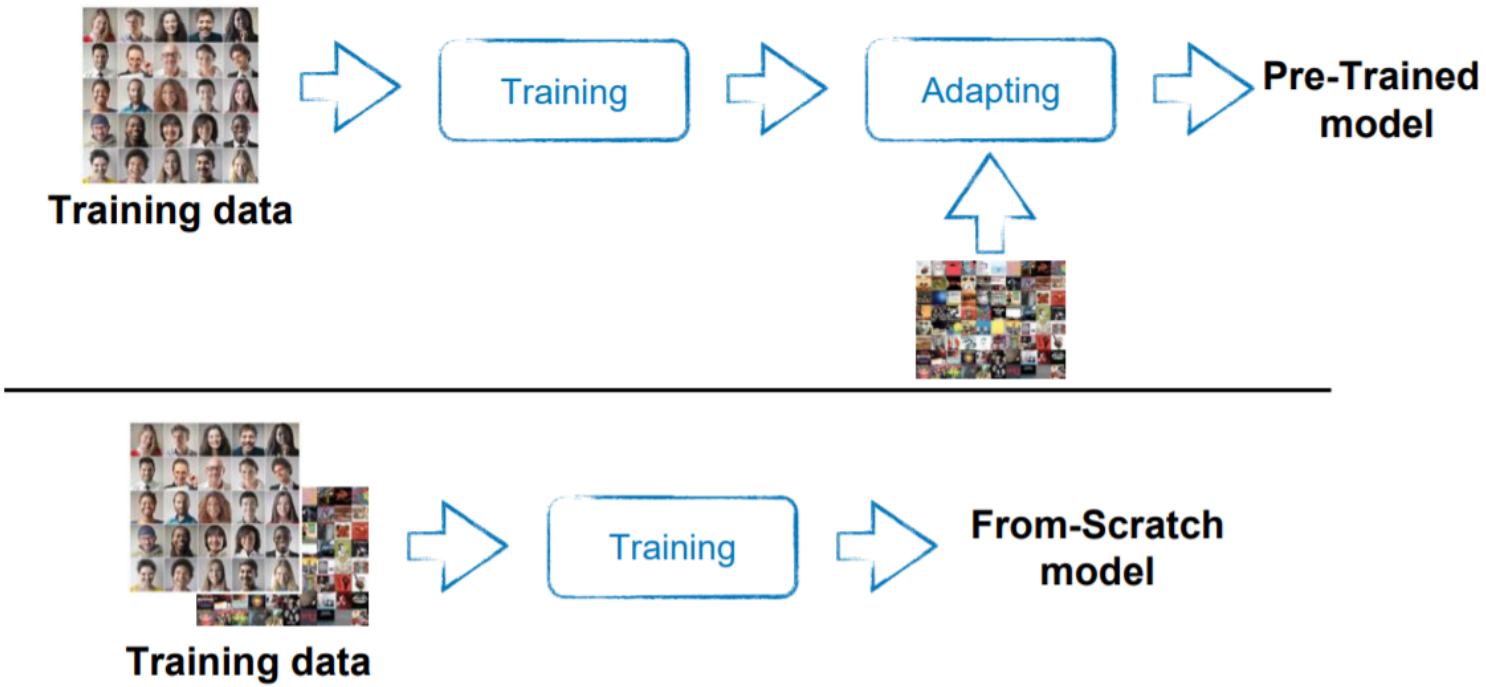
Figure 5: An example image from the trigger set. The label that was assigned to this image was "automobile".

$$\Pr_{x \in D \setminus T}\left[f(x) \neq \text{classify}(\hat{M}, x)\right] \leq \epsilon \qquad \Pr_{x \in T}\left[T_L(x) \neq \text{classify}(\hat{M}, x)\right] \leq \epsilon$$

Backdoors in Neural Networks

Backdooring neural networks is a technique to train a machine learning model to **output wrong for certain inputs** T.

- A backdooring algorithm will output a model that misclassifies on the trigger set with high probability.



Watermarking From Backdooring Algorithms

A watermarking scheme is split into three algorithms: (KeyGen, Mark, Verify)

Watermarking From Backdooring Algorithms

A watermarking scheme is split into three algorithms: (**KeyGen**, Mark, Verify)

$\texttt{KeyGen}()$:

1. Run $(T, T_L) = \mathsf{b} \leftarrow \texttt{SampleBackdoor}(\mathcal{O}^f)$ where $T = \{t^{(1)}, \ldots, t^{(n)}\}$ and $T_L = \{T_L^{(1)}, \ldots, T_L^{(n)}\}$.
2. Sample $2n$ random strings $r_t^{(i)}, r_L^{(i)} \leftarrow \{0,1\}^n$ and generate $2n$ commitments $\{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$ where $c_t^{(i)} \leftarrow \texttt{Com}(t^{(i)}, r_t^{(i)})$, $c_L^{(i)} \leftarrow \texttt{Com}(T_L^{(i)}, r_L^{(i)})$.
3. Set $\mathsf{mk} \leftarrow (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$, $\mathsf{vk} \leftarrow \{c_t^{(i)}, c_L^{(i)}\}_{i \in [n]}$ and return $(\mathsf{mk}, \mathsf{vk})$.

Watermarking From Backdooring Algorithms

A watermarking scheme is split into three algorithms: (KeyGen, **Mark**, Verify)

$\texttt{Mark}(M, \mathsf{mk})$:

1. Let $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i \in [n]})$.
2. Compute and output $\hat{M} \leftarrow \texttt{Backdoor}(\mathcal{O}^f, \mathsf{b}, M)$.

Watermarking From Backdooring Algorithms

A watermarking scheme is split into three algorithms: (KeyGen, Mark, **Verify**)

$\texttt{Verify}(\mathsf{mk}, \mathsf{vk}, M)$:

1. Let $\mathsf{mk} = (\mathsf{b}, \{r_t^{(i)}, r_L^{(i)}\}_{i\in[n]})$, $\mathsf{vk} = \{c_t^{(i)}, c_L^{(i)}\}_{i\in[n]}$. For $\mathsf{b} = (T, T_L)$ test if $\forall t^{(i)} \in T : T_L^{(i)} \neq f(t^{(i)})$. If not, then output 0.

2. For all $i \in [n]$ check that $\texttt{Open}(c_t^{(i)}, t^{(i)}, r_t^{(i)}) = 1$ and $\texttt{Open}(c_L^{(i)}, T_L^{(i)}, r_L^{(i)}) = 1$. Otherwise output 0.

3. For all $i \in [n]$ test that $\texttt{Classify}(t^{(i)}, M) = T_L^{(i)}$. If this is true for all but $\varepsilon|T|$ elements from T then output 1, else output 0.

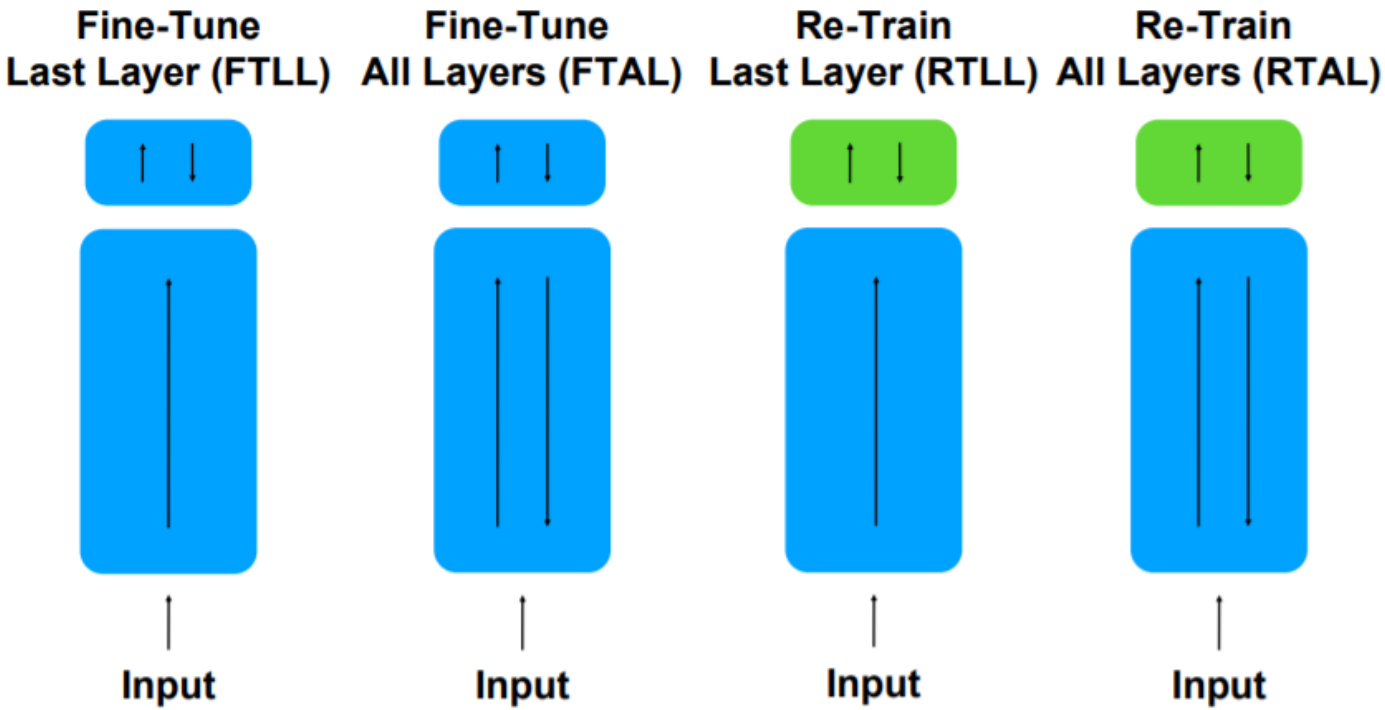
Results - Functionality Preserving

Notice that, the trigger set not classified correctly without embedding of WM.

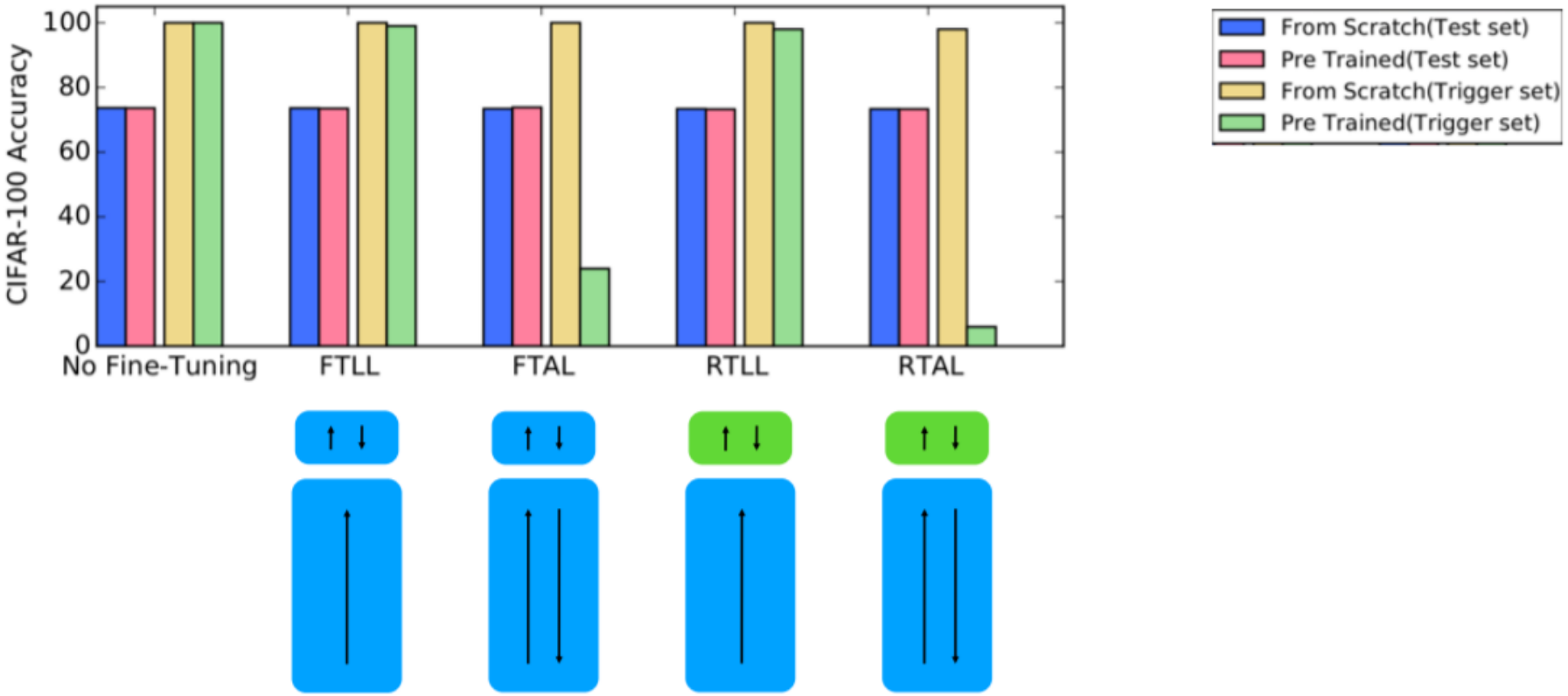
Model	Test-set acc.	Trigger-set acc.
CIFAR-10		
NO-WM	93.42	7.0
FROMSCRATCH	93.81	100.0
PRETRAINED	93.65	100.0
CIFAR-100		
NO-WM	74.01	1.0
FROMSCRATCH	73.67	100.0
PRETRAINED	73.62	100.0

Table 1: Classification accuracy for CIFAR-10 and CIFAR-100 datasets on the test set and trigger set.

Results - Unremovability



Results - Unremovability



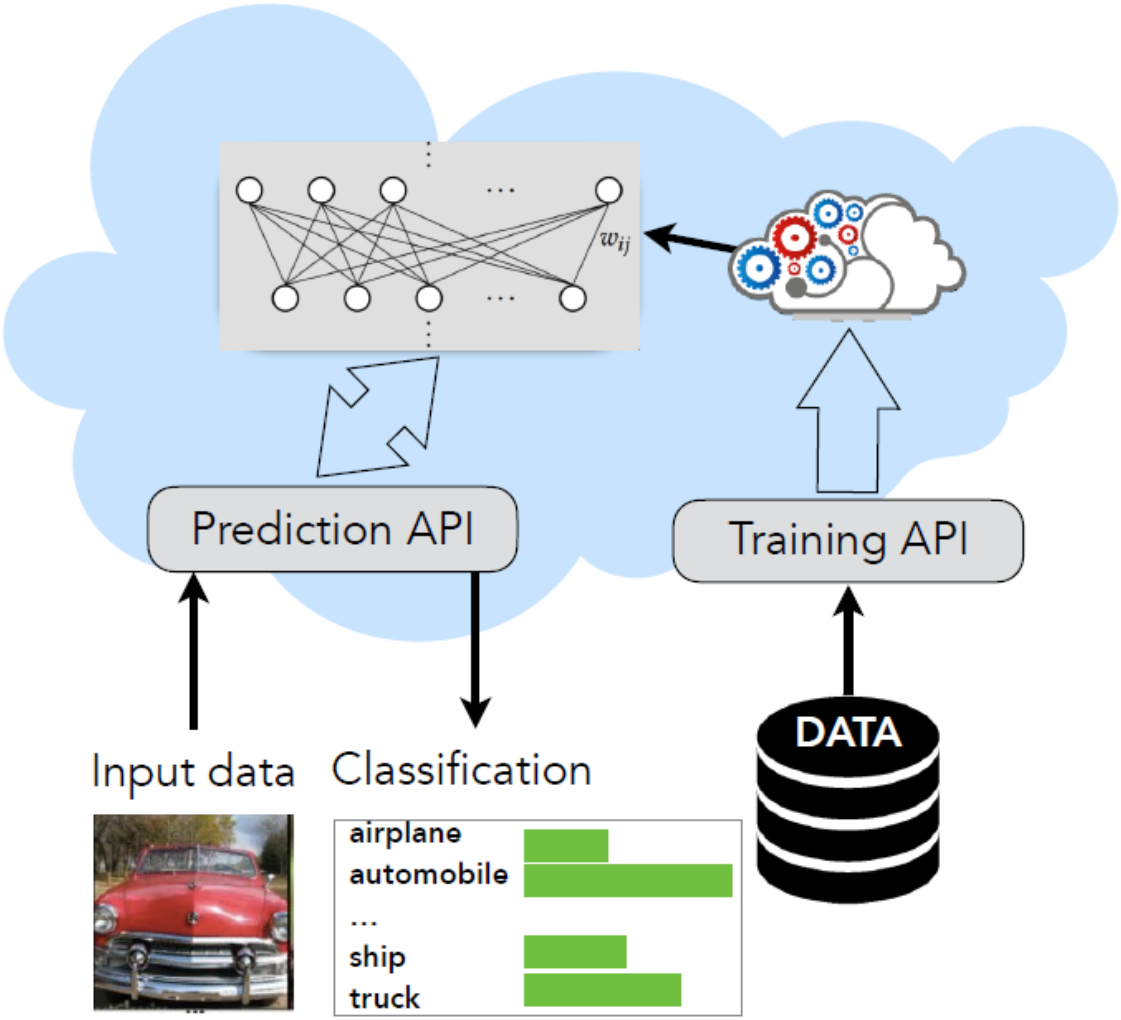
Privacy Risks

Machine Learning as a Service

Machine Learning as a Service

Google Cloud Platform

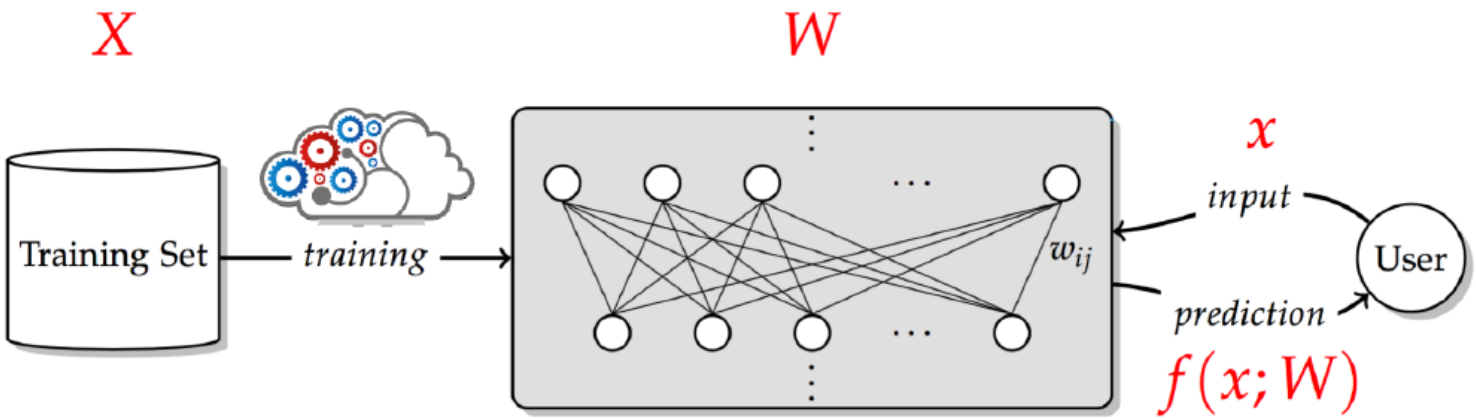
amazon web services™



(Shokri, 2020)

Privacy Risks in Machine Learning

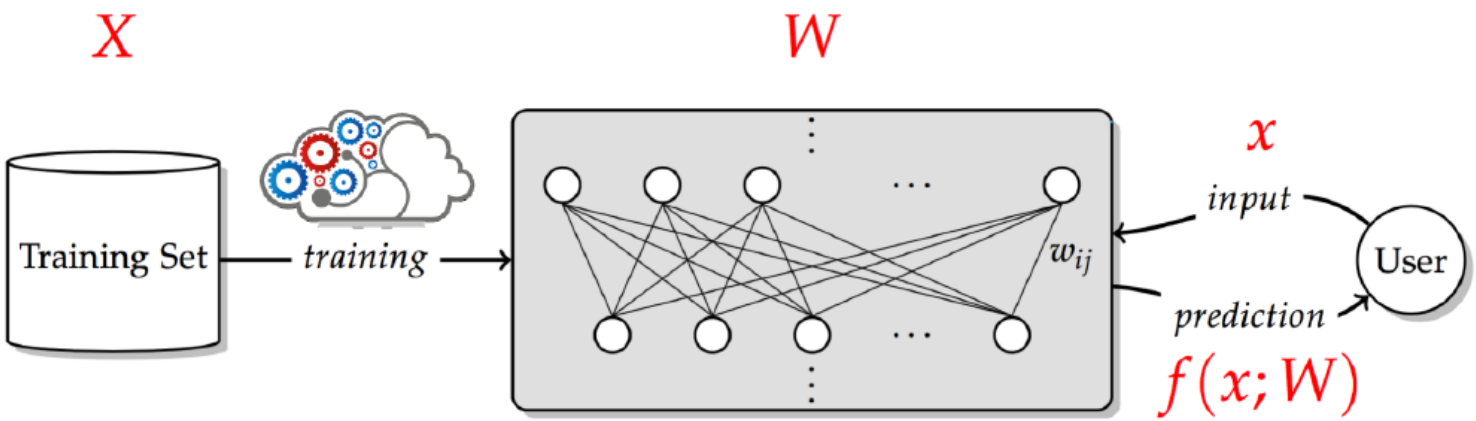
- What is training data leakage? Inferring information about members of X, beyond what can be learned about its underlying distribution.



(Shokri, 2020)

Privacy Risks in Machine Learning

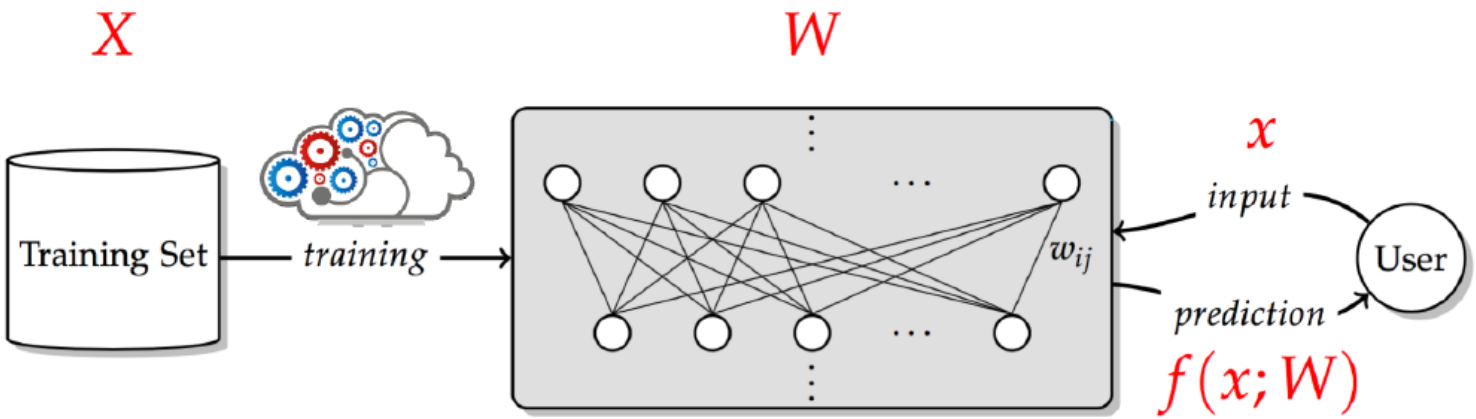
- What is training data leakage? Inferring information about members of X, beyond what can be learned about its underlying distribution.
- Valuable things: Training set X, user's data x, parameters W, prediction, etc.



(Shokri, 2020)

Privacy Risks in Machine Learning

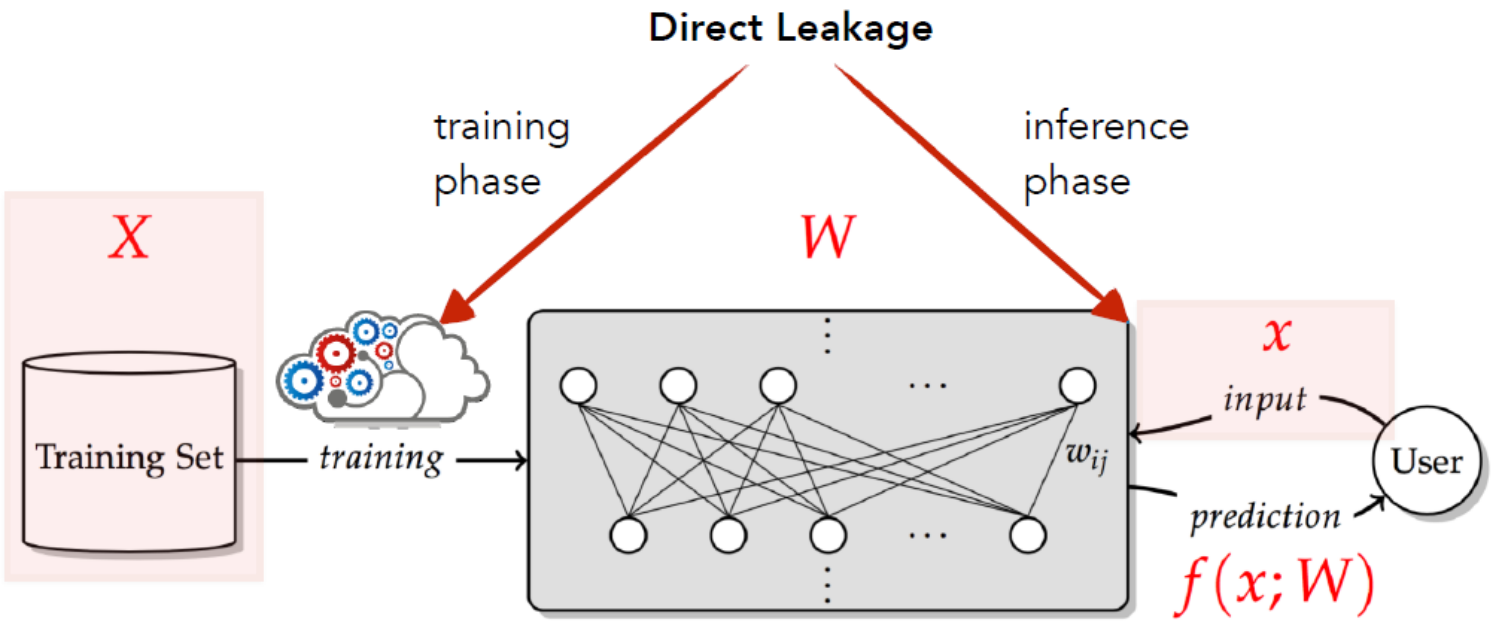
- What is training data leakage? Inferring information about members of X, beyond what can be learned about its underlying distribution.
- Valuable things: Training set X, user's data x, parameters W, prediction, etc.
- Adversary: malicious cloud, malicious user, the malicious data owner, etc.



(Shokri, 2020)

Privacy Risks in Machine Learning

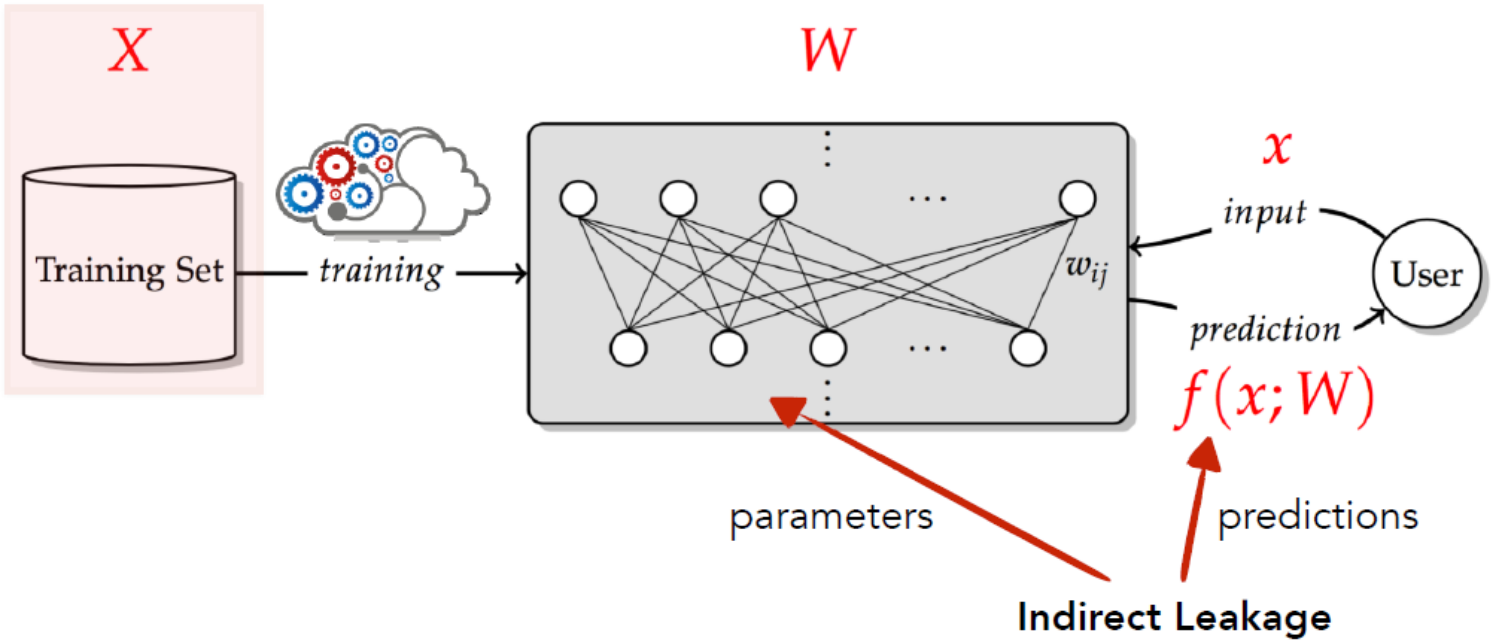
- How to prevent direct leakage? Secure multi-party computation, federated learning, homomorphic encryption, trusted hardware



(Shokri, 2020)

Privacy Risks in Machine Learning

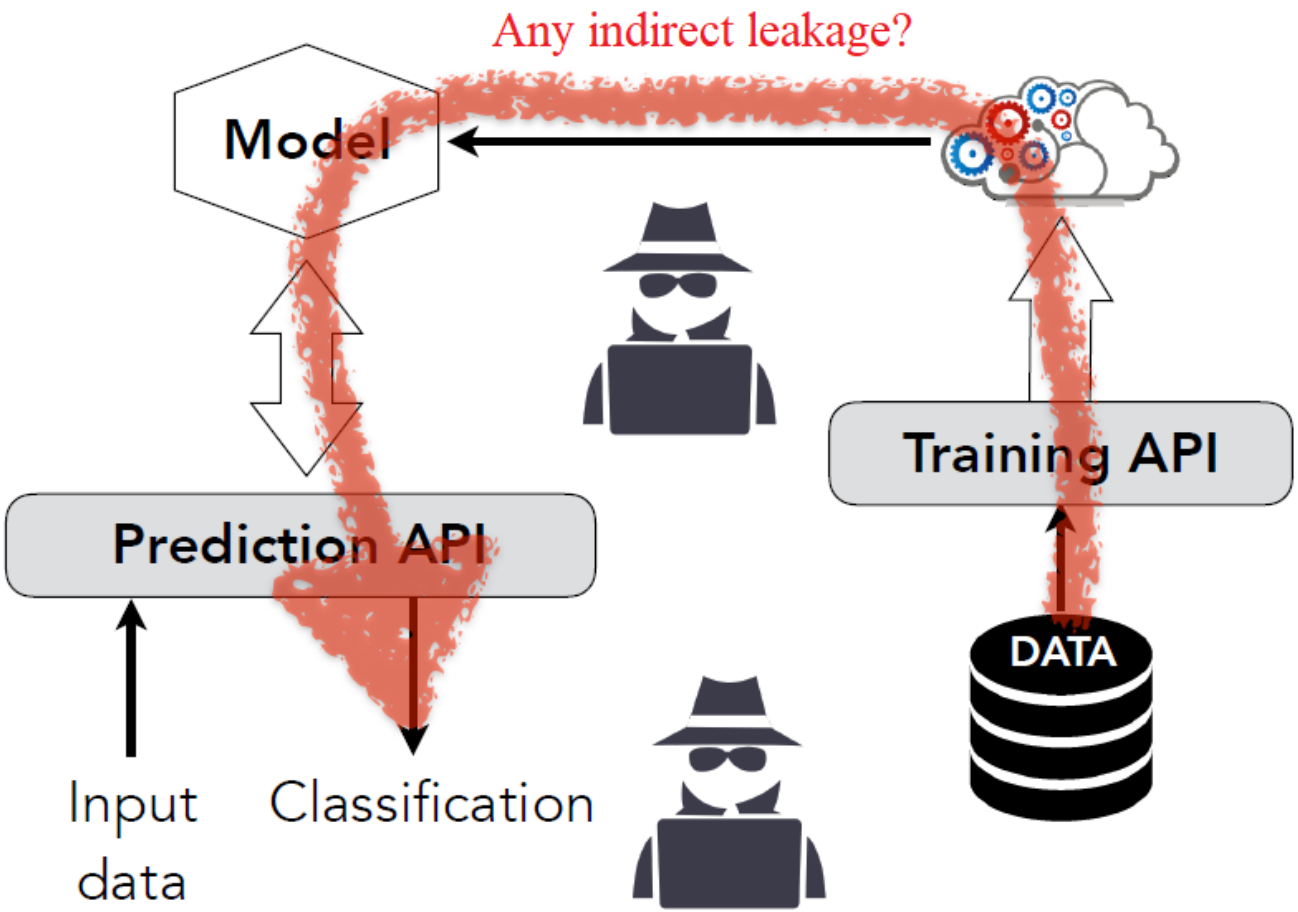
- How to prevent direct leakage? Secure multi-party computation, federated learning, homomorphic encryption, trusted hardware
- How to mitigate the indirect leakage? Differential privacy



(Shokri, 2020)

Membership Inference Attack

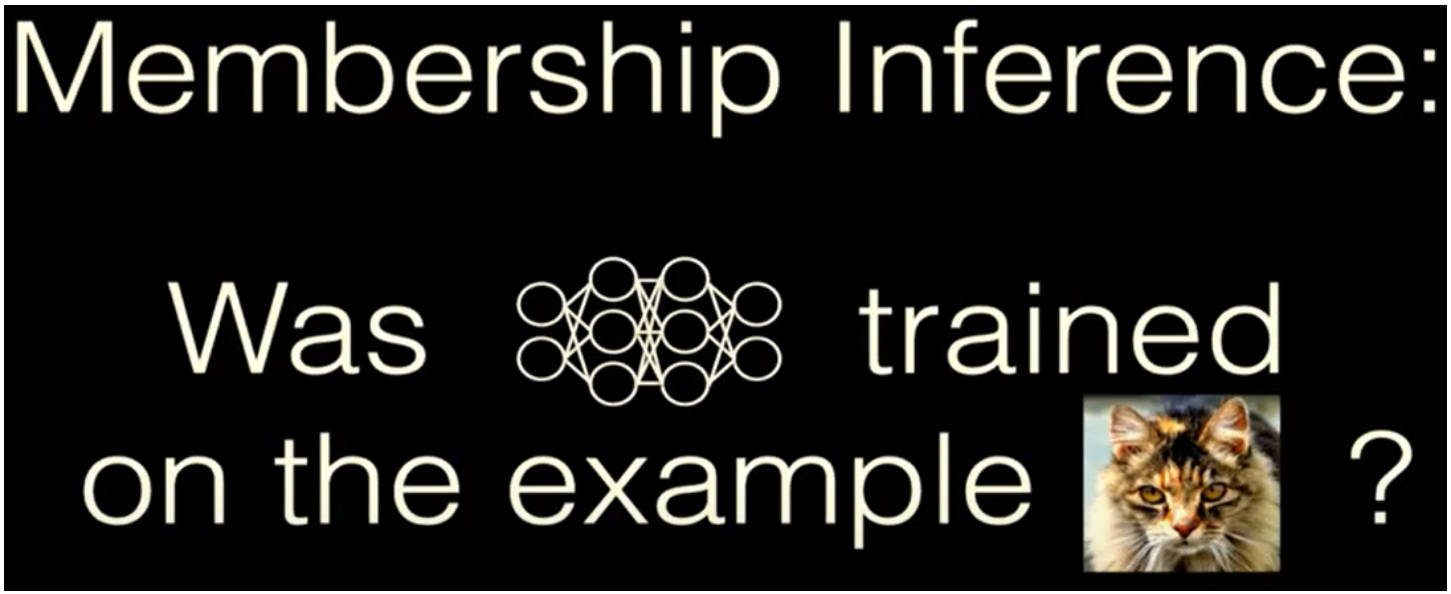
- Given a model, can an adversary infer whether data point x is part of its training set?



(Shokri, 2020)

Membership Inference Attack

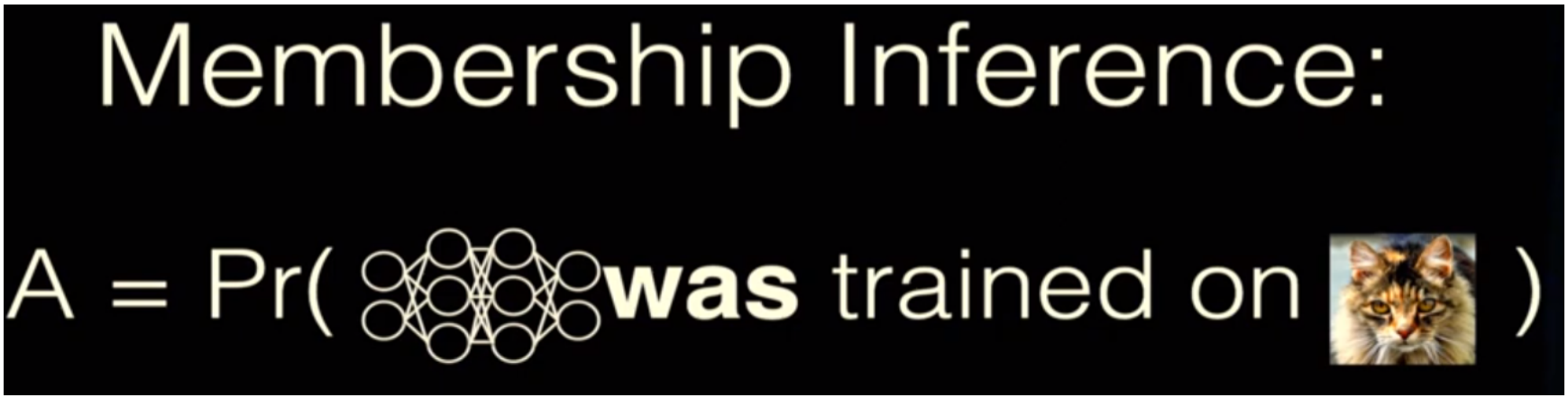
- Given a model, can an adversary infer whether data point x is part of its training set?



(Carlini, 2022)

Membership Inference Attack

- Given a model, can an adversary infer whether data point x is part of its training set?



(Carlini, 2022)

Model Inversion Attacks

- The attack uses a trained classifier in order to extract representations of the training data.

Original face image (right) and restored one through model inversion (left)

Model Inversion Attacks

- The attack uses a trained classifier in order to extract representations of the training data.

Algorithm 1 Inversion attack for facial recognition models.

function MI-FACE($label, \alpha, \beta, \gamma, \lambda$)
  $c(\mathbf{x}) \stackrel{\text{def}}{=} 1 - \tilde{f}_{label}(\mathbf{x}) + \text{AUXTERM}(\mathbf{x})$
  $\mathbf{x}_0 \leftarrow \mathbf{0}$
  for $i \leftarrow 1 \dots \alpha$ **do**
    $\mathbf{x}_i \leftarrow \text{PROCESS}(\mathbf{x}_{i-1} - \lambda \cdot \nabla c(\mathbf{x}_{i-1}))$
    if $c(\mathbf{x}_i) \geq \max(c(\mathbf{x}_{i-1}), \dots, c(\mathbf{x}_{i-\beta}))$ **then**
      break
    if $c(\mathbf{x}_i) \leq \gamma$ **then**
      break
  return $[\arg\min_{\mathbf{x}_i}(c(\mathbf{x}_i)), \min_{\mathbf{x}_i}(c(\mathbf{x}_i))]$

Generative Sequence Models

WHEN YOU TRAIN PREDICTIVE MODELS ON INPUT FROM YOUR USERS, IT CAN LEAK INFORMATION IN UNEXPECTED WAYS.

https://xkcd.com/2169/

Generative Sequence Models

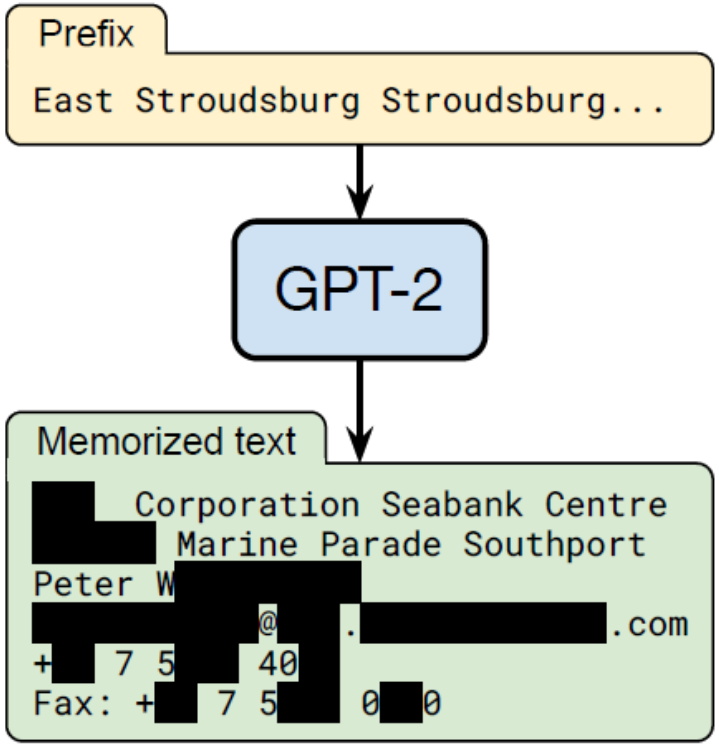


Figure 1: **Our extraction attack.** Given query access to a neural network language model, we extract an individual person's name, email address, phone number, fax number, and physical address. The example in this figure shows information that is all accurate so we redact it to protect privacy.

(*Extracting Training Data from Large Language Models*, Carlini, 2021)

Membership Inference Attacks

Membership Inference Attacks

Membership Inference Attacks Against Machine Learning Models

Reza Shokri
Cornell Tech
shokri@cornell.edu

Marco Stronati*
INRIA
marco@stronati.org

Congzheng Song
Cornell
cs2296@cornell.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

Abstract

membership inference attack

- Given a machine learning model and a record, determine whether this record was used as part of the model's training dataset or not.

Threat Model

- It is investigated in the most difficult setting, where the adversary's access to the model is limited to **black-box** queries that return the model's output on a given input.

Approach

- Train an **inference model** to recognize **differences in the target model's predictions** on the inputs that it trained on versus the inputs that it did not train on.

Threat Model

- The adversary has query access to the model and can obtain the model's **prediction vector** on any data record.

- The attacker may have some background knowledge about the population from which the target model's training dataset was drawn.
  - For example, he may have **independently drawn samples from the population**, disjoint from the target model's training dataset.

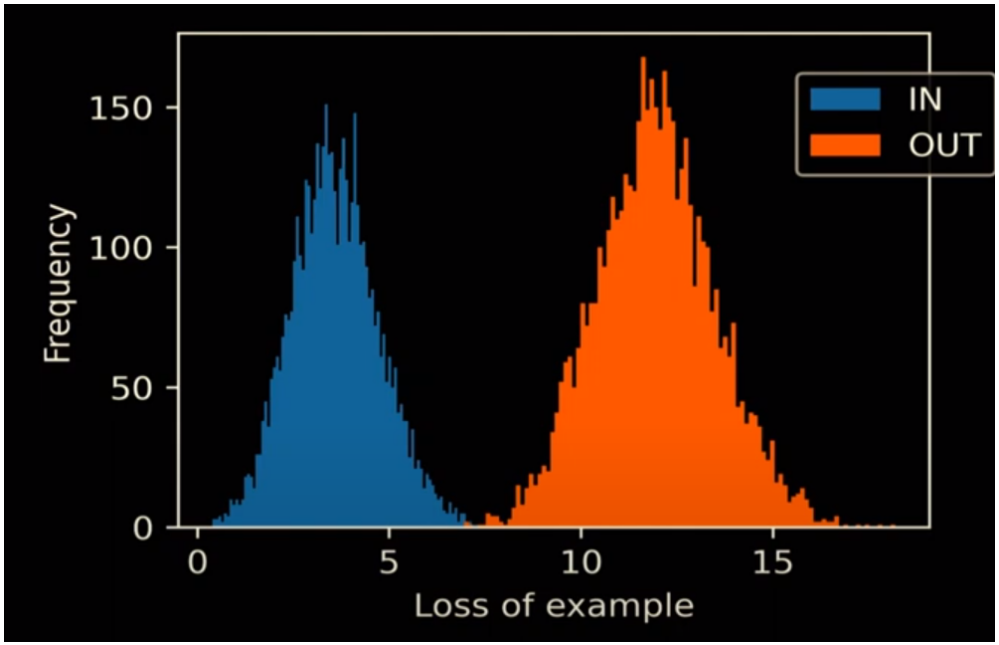
Overview of the attack

Membership inference attack exploits the observation that

- Machine learning models often **behave differently** on the data that they **were trained** on versus the data that they **see for the first time**.

The objective of the attacker is

- Construct an **attack model** that can **recognize such differences** in the target model's behavior and use them to distinguish members from non-members of the target model's training dataset based solely on the target model's output.



(Carlini, 2022)

Overview of the attack

Let f_{target} be the target model, and let D^{train}_{target} be its private training dataset which contains labeled data records $(x^{\{i\}}, y^{\{i\}})_{target}$.

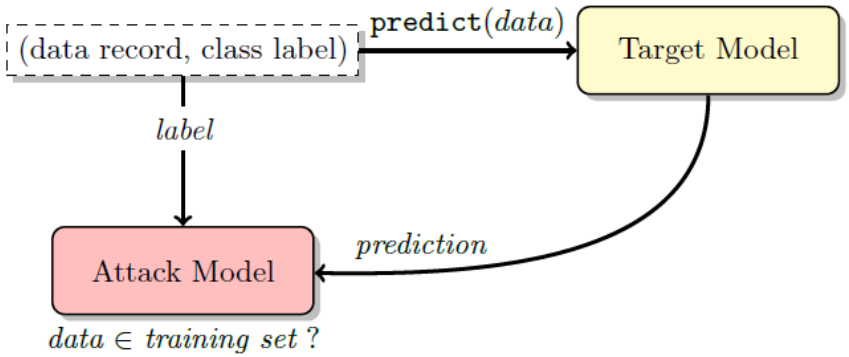


Fig. 1: Membership inference attack in the black-box setting. The

Overview of the attack

Let f_{target} be the target model, and let D_{target}^{train} be its private training dataset which contains labeled data records $(x^{\{i\}}, y^{\{i\}})_{target}$.

Let $f_{attack}()$ be the attack model, and its input x_{attack} is composed of

- A correctly labeled record
- A prediction vector of size c_{target}

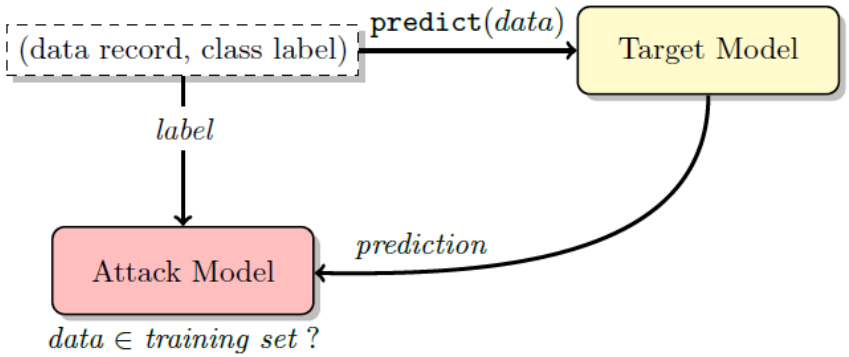


Fig. 1: Membership inference attack in the black-box setting. The

Overview of the attack

Let f_{target} be the target model, and let D_{target}^{train} be its private training dataset which contains labeled data records $(x^{\{i\}}, y^{\{i\}})_{target}$.

Let $f_{attack}()$ be the attack model, and its input x_{attack} is composed of

- A correctly labeled record
- A prediction vector of size c_{target}

Since the goal of the attack is decisional membership inference, the attack model is a **binary classifier** with two output classes, "in" and "out" or $Pr\{(x, y) \in D_{target}^{train}\}$.

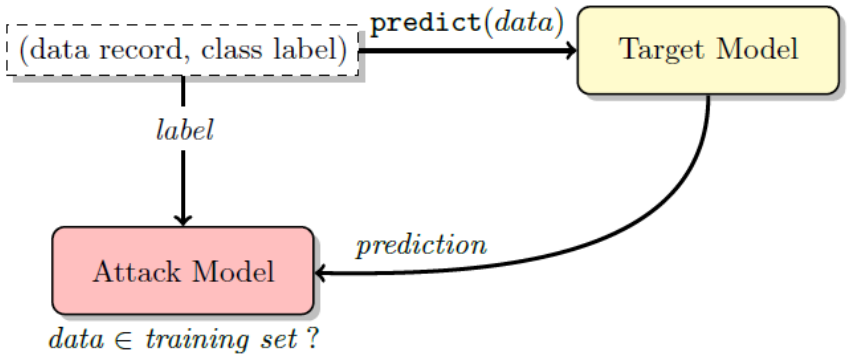


Fig. 1: Membership inference attack in the black-box setting. The

Shadow Models

To create the attack model's training set, the attack uses **shadow models**.

- Shadow models are created to **behave similarly to the target model**.
- The adversary creates k shadow models $f^i_{shadow}()$.
  - Each shadow model i is trained on a dataset $D^{train}_{shadow^i}$.

Shadow Models

To create the attack model's training set, the attack uses **shadow models**.

- Shadow models are created to **behave similarly to the target model**.
- The adversary creates k shadow models $f^i_{shadow}()$.
  - Each shadow model i is trained on a dataset $D^{train}_{shadow^i}$.
- In contrast to the target model, **we know the ground truth** for each shadow model, i.e., whether a given record was in its training dataset or not.

Shadow Models

To create the attack model's training set, the attack uses **shadow models**.

- Shadow models are created to **behave similarly to the target model**.
- The adversary creates k shadow models $f^i_{shadow}()$.
  - Each shadow model i is trained on a dataset $D^{train}_{shadow^i}$.
- In contrast to the target model, **we know the ground truth** for each shadow model, i.e., whether a given record was in its training dataset or not.
- Approach
  - We query each shadow model with its own training dataset and with a disjoint test set of the same size.
  - The outputs on the training dataset are labeled "in", the rest are labeled "out".
  - The attacker has a dataset D_{train} of records, the corresponding outputs of the shadow models, and the in/out labels.
  - Split D_{train} attack into c_{target} partitions, each associated with a different class label.
  - For each label y, train a separate attack model.

Shadow Models

To create the attack model's training set, the attack uses **shadow models**.

- Shadow models are created to **behave similarly to the target model**.
- The adversary creates k shadow models $f^i_{shadow}()$.
  - Each shadow model i is trained on a dataset $D^{train}_{shadow^i}$.
- In contrast to the target model, **we know the ground truth** for each shadow model, i.e., whether a given record was in its training dataset or not.
- Approach
  - We query each shadow model with its own training dataset and with a disjoint test set of the same size.
  - The outputs on the training dataset are labeled "in", the rest are labeled "out".
  - The attacker has a dataset D_{train} of records, the corresponding outputs of the shadow models, and the in/out labels.
  - Split D_{train} attack into c_{target} partitions, each associated with a different class label.
  - For each label y, train a separate attack model.

The shadow models must be **trained in a similar way** to the target model.

- This is easy if the target's training algorithm and model structure are known.
- In MLaaS, the attacker can **use exactly the same service** (e.g., Google Prediction API) to train the shadow model as was used to train the target model.

Training the attack model

The attack model training set D^{train}_{attack}

- For all $(x, y) \in D^{train}_{shadow^i}$, compute the prediction vector $\hat{y} = f^i_{shadow}(x)$ and add the record $((\hat{y}, y), in)$ to the attack training set D^{train}_{attack}.
- For all $(x, y) \in D^{test}_{shadow^i}$, compute the prediction vector $\hat{y} = f^i_{shadow}(x)$ and add the record $((\hat{y}, y), out)$ to the attack training set D^{train}_{attack}.

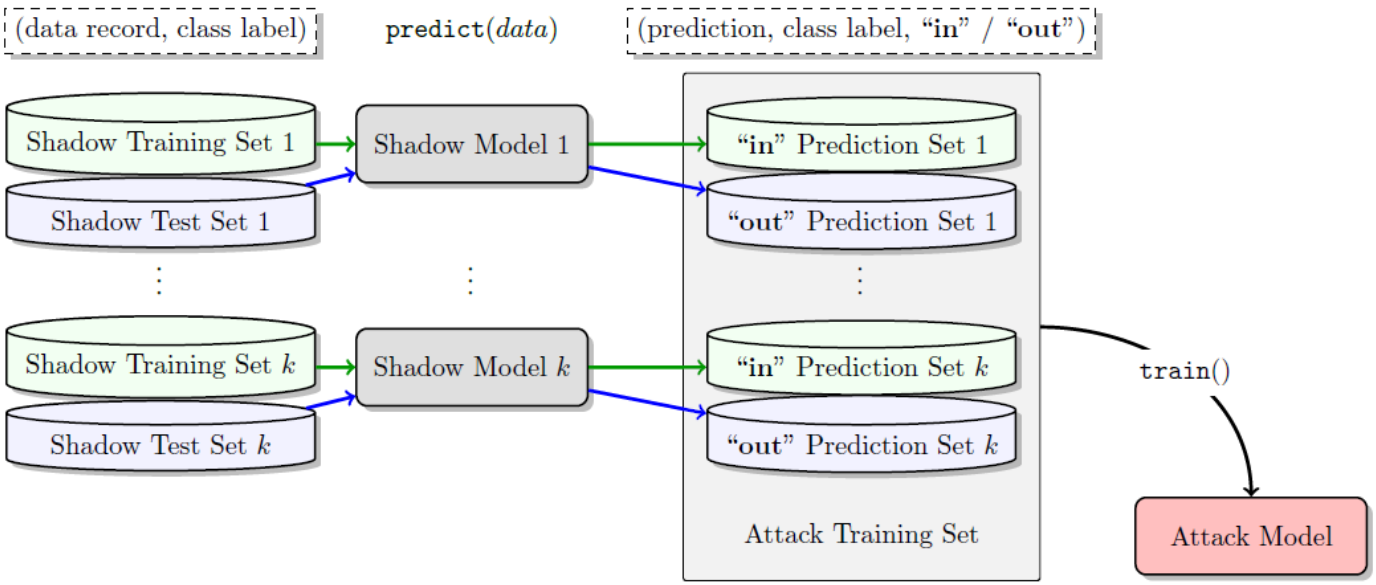


Fig. 3: Training the attack model on the inputs and outputs of the shadow models. For all records in the training dataset of a shadow model, we query the model and obtain the output. These output vectors are labeled "in" and added to the attack model's training dataset. We also query the shadow model with a test dataset disjoint from its training dataset. The outputs on this set are labeled "out" and also added to the attack model's training dataset. Having constructed a dataset that reflects the black-box behavior of the shadow models on their training and test datasets, we train a collection of c_{target} attack models, one per each output class of the target model.

Shadow Models

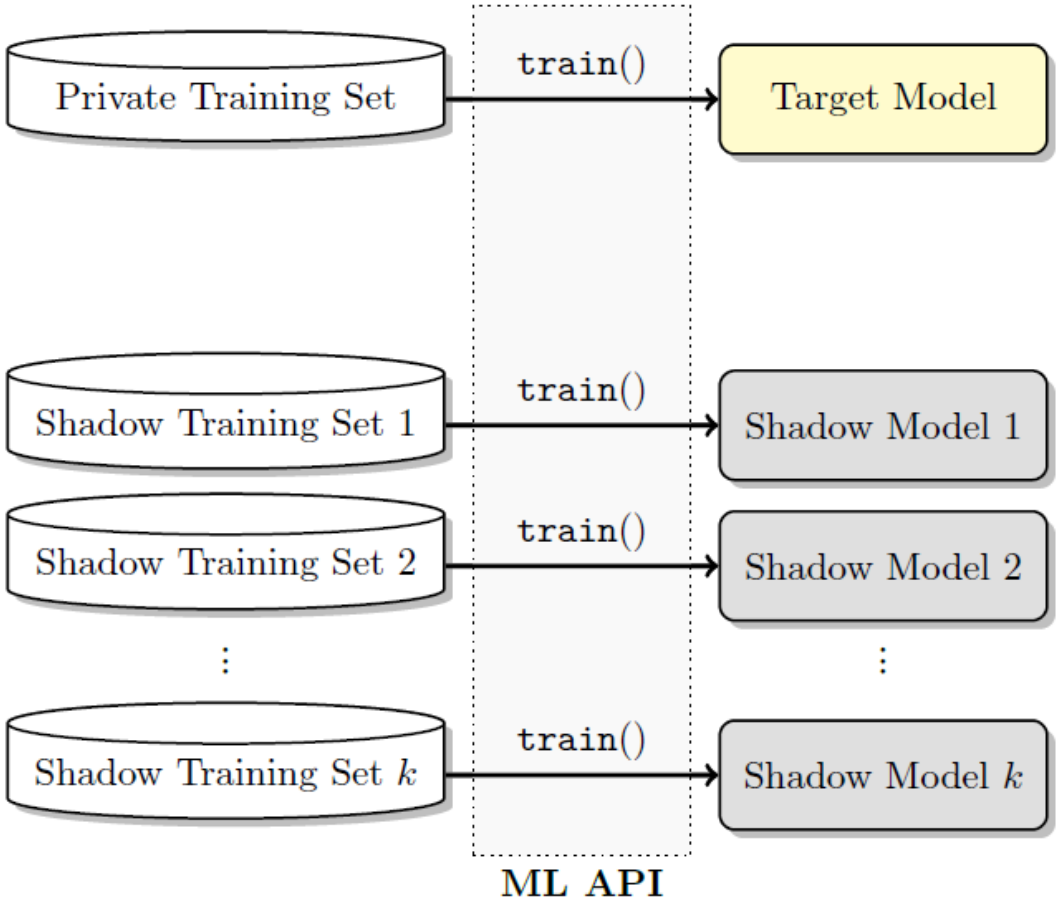


Fig. 2: Training shadow models using the same machine learning platform as was used to train the target model. The training datasets of the target and shadow models have the same format but are disjoint. The training datasets of the shadow models may overlap. All models' internal parameters are trained independently.

Generating training data for shadow models

Generating training data for shadow model

1. Model-based synthesis
2. Statistics-based synthesis
3. Noisy real data

Model-based synthesis

If the attacker does not have real training data nor any statistics about its distribution, he can generate synthetic training data for the shadow models **using the target model** itself.

Model-based synthesis

If the attacker does not have real training data nor any statistics about its distribution, he can generate synthetic training data for the shadow models **using the target model** itself.

- The intuition is that records that are classified by the target model with **high confidence** should be statistically similar to the target's training dataset.
- The synthesis process runs in two phases
  1. search, using a hill-climbing algorithm, the space of possible data records to find inputs that are classified by the target model with high confidence
  2. sample synthetic data from these records.

Model-based synthesis

If the attacker does not have real training data nor any statistics about its distribution, he can generate synthetic training data for the shadow models **using the target model** itself.

- The intuition is that records that are classified by the target model with **high confidence** should be statistically similar to the target's training dataset.
- The synthesis process runs in two phases
  1. search, using a hill-climbing algorithm, the space of possible data records to find inputs that are classified by the target model with high confidence
  2. sample synthetic data from these records.

It may **not work** if the inputs are **high resolution** images and the target model performs a complex image classification task.

Model-based synthesis

Algorithm 1 Data synthesis using the target model

1: **procedure** SYNTHESIZE(class : c)
2: $\mathbf{x} \leftarrow$ RANDRECORD( ) ▷ *initialize a record randomly*
3: $y_c^* \leftarrow 0$
4: $j \leftarrow 0$
5: $k \leftarrow k_{max}$
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27: **end procedure**

Model-based synthesis

Algorithm 1 Data synthesis using the target model

```
1:  procedure SYNTHESIZE(class : c)
2:      x ← RANDRECORD( )                          ▷ initialize a record randomly
3:      y*_c ← 0
4:      j ← 0
5:      k ← k_max
6:      for iteration = 1 ··· iter_max do
7:          y ← f_target(x)                        ▷ query the target model
8:          if y_c ≥ y*_c then                     ▷ accept the record
9:              if y_c > conf_min and c = arg max(y) then
10:                 if rand() < y_c then           ▷ sample
11:                     return x                   ▷ synthetic data
12:                 end if
13:             end if
14:
15:
16:
17:         else
18:
19:
20:
21:
22:
23:
24:         x ← RANDRECORD(x*, k)                  ▷ randomize k features
25:     end for
26:     return ⊥                                   ▷ failed to synthesize
27: end procedure
```

Model-based synthesis

Algorithm 1 Data synthesis using the target model

```
1:  procedure SYNTHESIZE(class : c)
2:      x ← RANDRECORD( )                          ▷ initialize a record randomly
3:      y*_c ← 0
4:      j ← 0
5:      k ← k_max
6:      for iteration = 1 ··· iter_max do
7:          y ← f_target(x)                        ▷ query the target model
8:          if y_c ≥ y*_c then                     ▷ accept the record
9:              if y_c > conf_min and c = arg max(y) then
10:                 if rand() < y_c then           ▷ sample
11:                     return x                   ▷ synthetic data
12:                 end if
13:             end if
14:             x* ← x
15:             y*_c ← y_c
16:             j ← 0
17:         else
18:
19:
20:
21:
22:
23:
24:         x ← RANDRECORD(x*, k)                  ▷ randomize k features
25:     end for
26:     return ⊥                                   ▷ failed to synthesize
27: end procedure
```

Model-based synthesis

Algorithm 1 Data synthesis using the target model

```
1:  procedure SYNTHESIZE(class : c)
2:      x ← RANDRECORD( )                          ▷ initialize a record randomly
3:      y*_c ← 0
4:      j ← 0
5:      k ← k_max
6:      for iteration = 1 ··· iter_max do
7:          y ← f_target(x)                        ▷ query the target model
8:          if y_c ≥ y*_c then                     ▷ accept the record
9:              if y_c > conf_min and c = arg max(y) then
10:                 if rand() < y_c then           ▷ sample
11:                     return x                   ▷ synthetic data
12:                 end if
13:             end if
14:             x* ← x
15:             y*_c ← y_c
16:             j ← 0
17:         else
18:             j ← j + 1
19:             if j > rej_max then                ▷ many consecutive rejects
20:                 k ← max(k_min, ⌈k/2⌉)
21:                 j ← 0
22:             end if
23:         end if
24:         x ← RANDRECORD(x*, k)                  ▷ randomize k features
25:     end for
26:     return ⊥                                   ▷ failed to synthesize
27: end procedure
```

Statistics-based synthesis

The attacker may have some **statistical information** about the population from which the target model's training data was drawn.

- Generating synthetic training records for the shadow models by **independently sampling the value of each feature** from its own marginal distribution.

Noisy real data

The attacker may have access to some data that is **similar to the target model's training data** and can be considered as a "noisy" version.

- Flipping the (binary) values of 10% or 20% randomly selected features, then training our shadow models on the resulting noisy dataset.

Evaluation

Datasets

- CIFAR-10 and CIFAR-100
- **Purchases**: The classification task is to predict the purchase style of a user given the 600-feature vector. 10000 randomly records are used to train the target model. {2; 10; 20; 50; 100} classes.
- Location: 446 binary features and 1600 training data into 30 classes.
- Texas hospital stays: The resulting dataset has 67330 records and 6170 binary features. 10000 randomly records are selected to train the target model.
- MNIST
- UCI Adult: This dataset includes 48842 records with 14 attributes. The (binary) classification task is to predict if a person makes over $50K a year. 10000 randomly records are used to train the target model.

Evaluation

Datasets

- CIFAR-10 and CIFAR-100
- **Purchases**: The classification task is to predict the purchase style of a user given the 600-feature vector. 10000 randomly records are used to train the target model. {2; 10; 20; 50; 100} classes.
- Location: 446 binary features and 1600 training data into 30 classes.
- Texas hospital stays: The resulting dataset has 67330 records and 6170 binary features. 10000 randomly records are selected to train the target model.
- MNIST
- UCI Adult: This dataset includes 48842 records with 14 attributes. The (binary) classification task is to predict if a person makes over $50K a year. 10000 randomly records are used to train the target model.

Target models

- Two implemented on cloud-based **machine learning as a service** platforms, Google Prediction API and Amazon ML
  - Purchases, Texas hospital stays, locations, Adult, and MNIST
- One implemented locally
  - Purchases, CIFAR10, and CIFAR100

Shadow Models

The number of shadow models is

- 100 for the CIFAR datasets
- 20 for the purchase dataset
- 10 for the Texas hospital stay dataset
- 60 for the location dataset
- 50 for the MNIST dataset
- 20 for the Adult dataset

Increasing the number of shadow models would **increase the accuracy** of the attack but also its **cost**.

Accuracy of the attack

The attack is evaluated by executing it on randomly reshuffled records from **the target's training and test datasets** and the have the same size.

- Hence the baseline accuracy is $\frac{1}{2}$.

Evaluation

- The test accuracy of our target neural-network models with the largest training datasets (15000 and 29540 records, respectively) is 60% and 20% for CIFAR-10 and CIFAR-100, respectively.
- Use real data to train shadow models

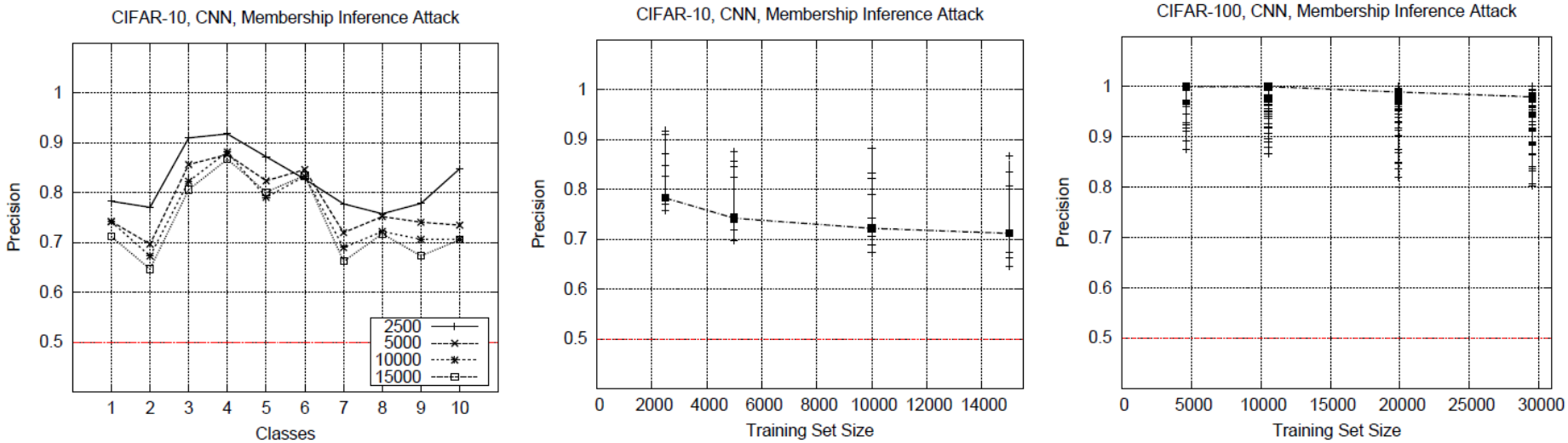
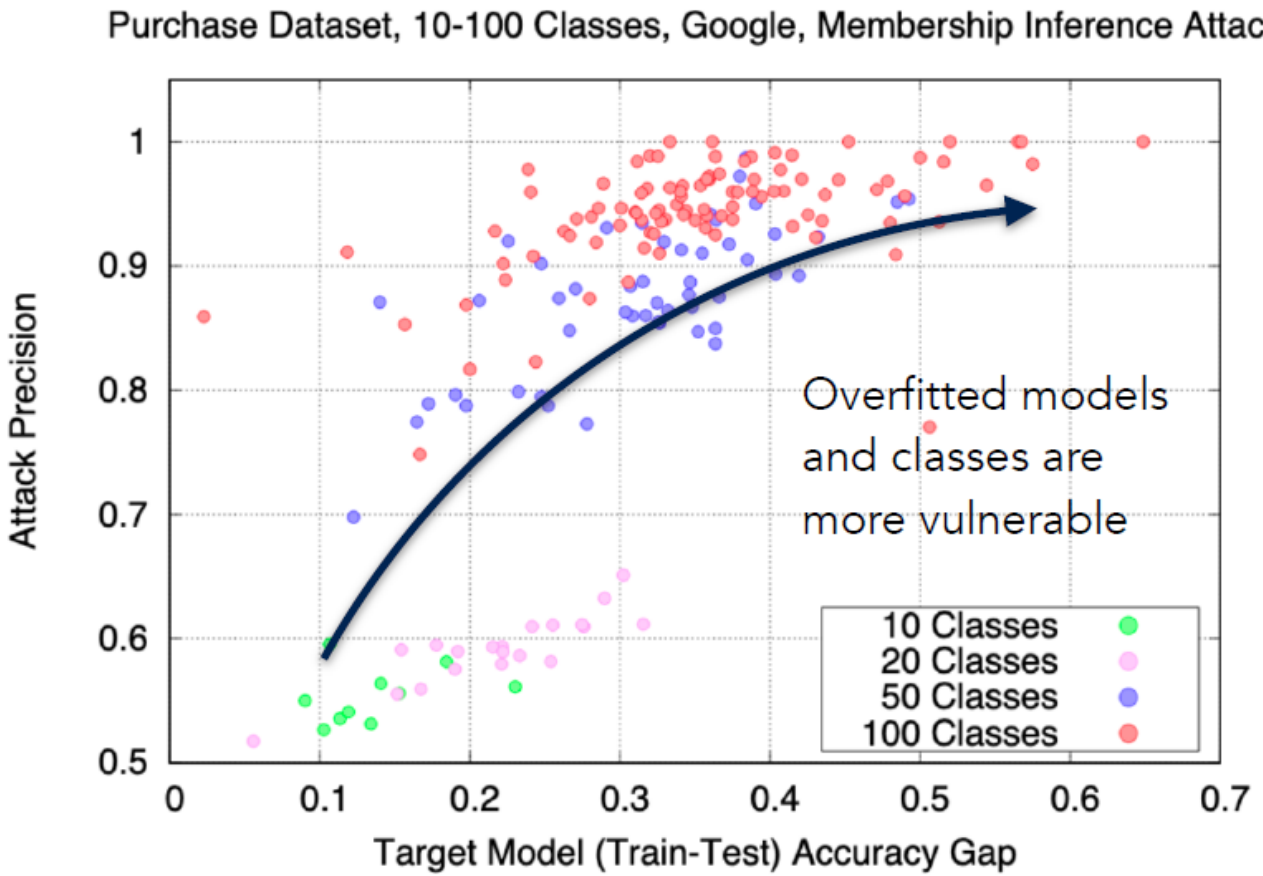


Fig. 4: Precision of the membership inference attack against neural networks trained on CIFAR datasets. The graphs show precision for different classes while varying the size of the training datasets. The median values are connected across different training set sizes. The median precision (from the smallest dataset size to largest) is $0.78, 0.74, 0.72, 0.71$ for CIFAR-10 and $1, 1, 0.98, 0.97$ for CIFAR-100. Recall is almost 1 for both datasets. The figure on the left shows the per-class precision (for CIFAR-10). Random guessing accuracy is 0.5.

Privacy Leakage due to Overfitting



Purchase Dataset

ML Platform	*Training*	*Test*
Google	0.999	0.656
Amazon (10,1e-6)	0.941	0.468
Amazon (100,1e-4)	1.00	0.504
Neural network	0.830	0.670

TABLE I: Training and test accuracy of the models constr
different ML-as-a-service platforms on the purchase datas
classes).

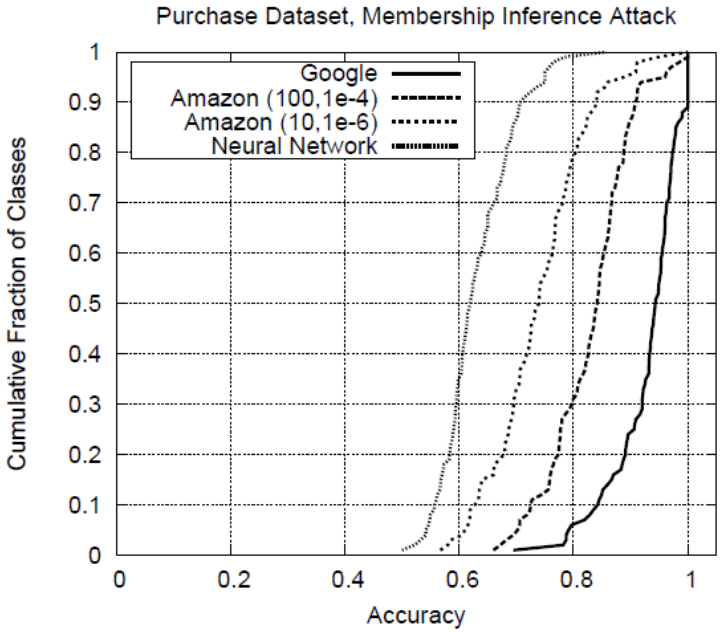


Fig. 7: Precision of the membership inference attack against models trained on the same datasets but using different platforms. The attack model is a neural network.

Overfitting is not the only factor that causes a model to be vulnerable to membership inference. The structure and type of the model also contribute to the problem.

Purchase Dataset

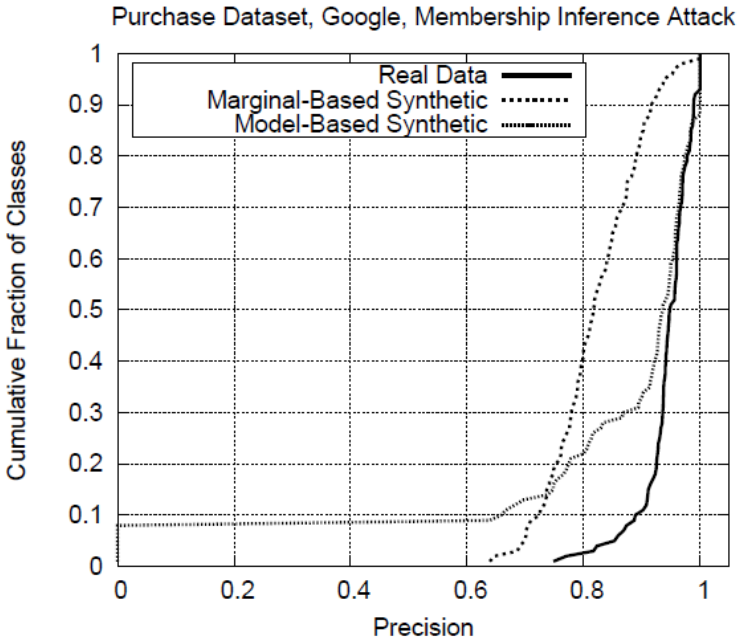


Fig. 9: Empirical CDF of the precision of the membership inference attack against the Google-trained model for the purchase dataset. Results are shown for different ways of generating training data for the shadow models (real, synthetic generated from the target model, synthetic generated from marginal statistics). Precision of the attack over all classes is 0.935 (real data), 0.795 (marginal-based synthetic data), and 0.896 (model-based synthetic data). The corresponding recall of the attack is 0.994, 0.991, and 0.526, respectively.

Overfitting vs. attack success

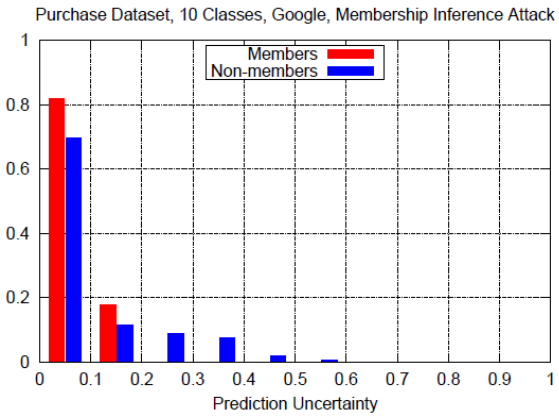
Dataset	*Training Accuracy*	*Testing Accuracy*	*Attack Precision*
Adult	0.848	0.842	0.503
MNIST	0.984	0.928	0.517
Location	1.000	0.673	0.678
Purchase (2)	0.999	0.984	0.505
Purchase (10)	0.999	0.866	0.550
Purchase (20)	1.000	0.781	0.590
Purchase (50)	1.000	0.693	0.860
Purchase (100)	0.999	0.659	0.935
TX hospital stays	0.668	0.517	0.657

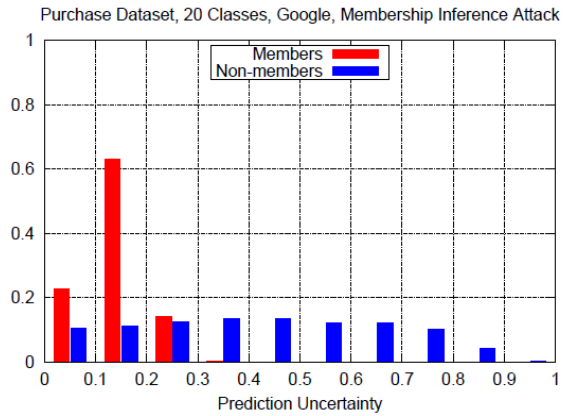
TABLE II: Accuracy of the Google-trained models and the corresponding attack precision.

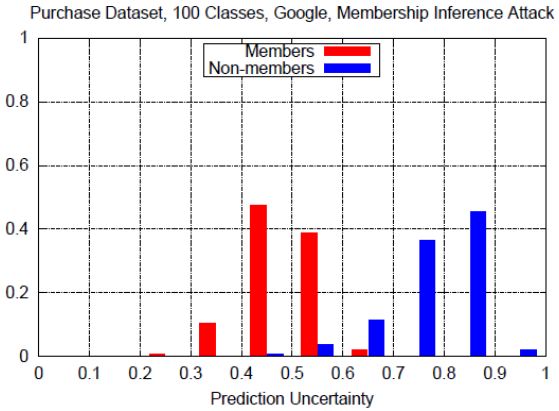
Classification uncertainty

Prediction uncertainty is the **normalized entropy** of the model's prediction vector

- $\frac{-1}{log(n)} \sum_i p_i \log(p_i)$ where p_i is the probability that the input belongs to class i, and n is the number of classes.







The accuracy of the target models with different mitigation techniques

Purchase dataset	*Testing Accuracy*	*Attack Total Accuracy*	*Attack Precision*	*Attack Recall*
No Mitigation	0.66	0.92	0.87	1.00
Top $k = 3$	0.66	0.92	0.87	0.99
Top $k = 1$	0.66	0.89	0.83	1.00
Top $k = 1$ label	0.66	0.66	0.60	0.99
Rounding $d = 3$	0.66	0.92	0.87	0.99
Rounding $d = 1$	0.66	0.89	0.83	1.00
Temperature $t = 5$	0.66	0.88	0.86	0.93
Temperature $t = 20$	0.66	0.84	0.83	0.86
L2 $\lambda = 1e-4$	0.68	0.87	0.81	0.96
L2 $\lambda = 1e-3$	0.72	0.77	0.73	0.86
L2 $\lambda = 1e-2$	0.63	0.53	0.54	0.52

- Label only: The attack can still exploit the mislabeling behavior of the target model because members and non-members of the training dataset are mislabeled differently (assigned to different wrong classes).

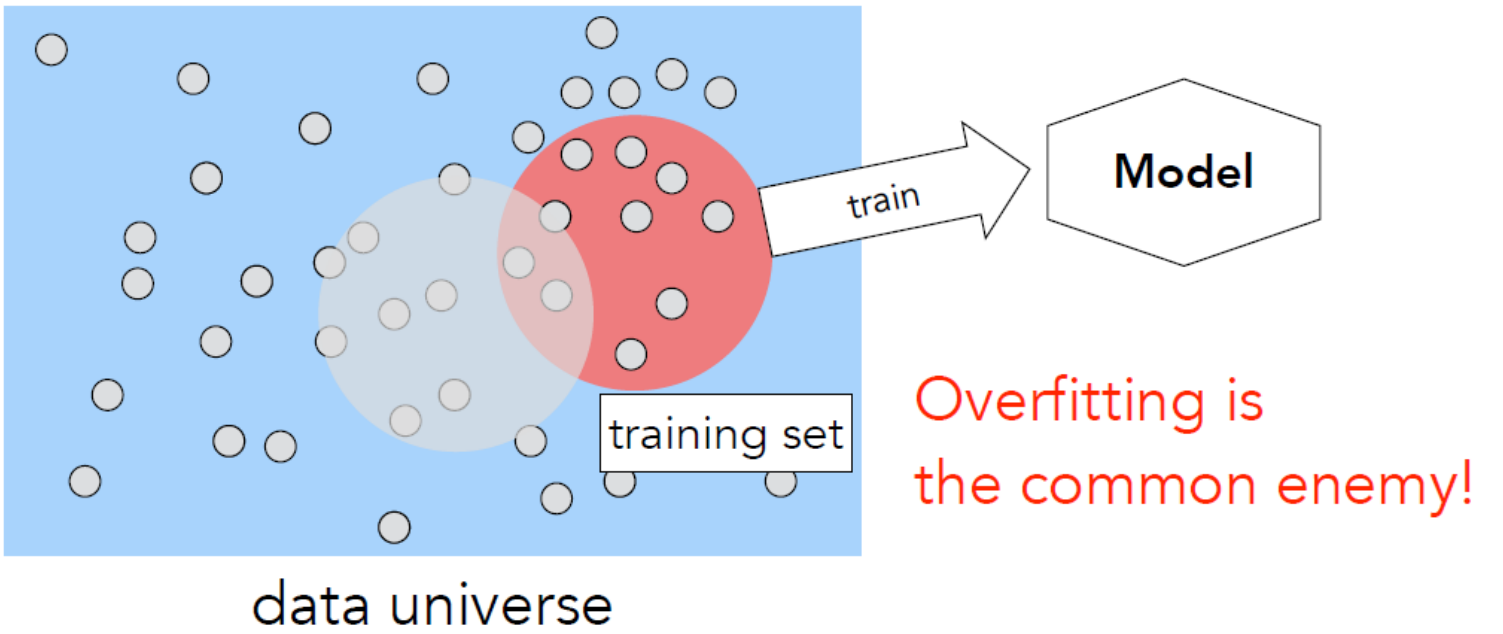
Privacy vs. Learning

Privacy

Does the model leak information about data in the training set?

Learning

Does the model generalize to data outside the training set?



Overfitting is the common enemy!

(Shokri, 2020)