MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples

Jinyuan Jia
ECE Department
Duke University
jinyuan.jia@duke.edu

Ahmed Salem
CISPA Helmholtz Center for Information Security
ahmed.salem@cispa.saarland

Michael Backes
CISPA Helmholtz Center for Information Security
backes@cispa.saarland

Yang Zhang
CISPA Helmholtz Center for Information Security
zhang@cispa.saarland

Neil Zhenqiang Gong
ECE Department
Duke University
neil.gong@duke.edu

ABSTRACT

In a membership inference attack, an attacker aims to infer whether a data sample is in a target classifier's training dataset or not. Specifically, given a black-box access to the target classifier, the attacker trains a binary classifier, which takes a data sample's confidence score vector predicted by the target classifier as an input and predicts the data sample to be a member or non-member of the target classifier's training dataset. Membership inference attacks pose severe privacy and security threats to the training dataset. Most existing defenses leverage differential privacy when training the target classifier or regularize the training process of the target classifier. These defenses suffer from two key limitations: 1) they do not have formal utility-loss guarantees of the confidence score vectors, and 2) they achieve suboptimal privacy-utility tradeoffs.

In this work, we propose *MemGuard*, the first defense with formal utility-loss guarantees against black-box membership inference attacks. Instead of tampering the training process of the target classifier, MemGuard adds noise to each confidence score vector predicted by the target classifier. Our key observation is that attacker uses a classifier to predict member or non-member and classifier is vulnerable to *adversarial examples*. Based on the observation, we propose to add a carefully crafted noise vector to a confidence score vector to turn it into an adversarial example that misleads the attacker's classifier. Specifically, MemGuard works in two phases. In Phase I, MemGuard finds a carefully crafted noise vector that can turn a confidence score vector into an adversarial example, which is likely to mislead the attacker's classifier to make a random guessing at member or non-member. We find such carefully crafted noise vector via a new method that we design to incorporate the unique utility-loss constraints on the noise vector. In Phase II, MemGuard adds the noise vector to the confidence score vector with a certain probability, which is selected to satisfy a given utility-loss budget on the confidence score vector. Our experimental results on three datasets show that MemGuard can effectively defend against membership inference attacks and achieve better privacy-utility tradeoffs than existing defenses. Our work is the first one to show that adversarial examples can be used as defensive mechanisms to defend against membership inference attacks.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363201

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Machine learning**;

KEYWORDS

Membership inference attacks; adversarial examples; privacy-preserving machine learning

ACM Reference Format:
Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. 2019. MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3319535.3363201

1 INTRODUCTION

Machine learning (ML) is transforming many aspects of our society. We consider a model provider deploys an ML classifier (called *target classifier*) as a black-box software or service, which returns a *confidence score vector* for a query data sample from a user. The confidence score vector is a probability distribution over the possible labels and the label of the query data sample is predicted as the one that has the largest confidence score. Multiple studies have shown that such black-box ML classifier is vulnerable to *membership inference attacks* [43, 56, 58, 59]. Specifically, an attacker trains a binary classifier, which takes a data sample's confidence score vector predicted by the target classifier as an input and predicts whether the data sample is a *member* or *non-member* of the target classifier's training dataset. Membership inference attacks pose severe privacy and security threats to ML. In particular, in application scenarios where the training dataset is sensitive (e.g., biomedical records and location traces), successful membership inference leads to severe privacy violations. For instance, if an attacker knows her victim's data is used to train a medical diagnosis classifier, then the attacker can directly infer the victim's health status. Beyond privacy,

membership inference also damages the model provider's intellectual property of the training dataset as collecting and labeling the training dataset may require lots of resources.

Therefore, defending against membership inference attacks is an urgent research problem and multiple defenses [42, 56, 58] have been explored. A major reason why membership inference attacks succeed is that the target classifier is overfitted. As a result, the confidence score vectors predicted by the target classifier are distinguishable for members and non-members of the training dataset. Therefore, state-of-the-art defenses [42, 56, 58] essentially regularize the training process of the target classifier to reduce overfitting and the gaps of the confidence score vectors between members and non-members of the training dataset. For instance, L_2 regularization [58], min-max game based adversarial regularization [42], and dropout [56] have been explored to regularize the target classifier. Another line of defenses [1, 6, 12, 24, 30, 60, 66, 70] leverage differential privacy [13] when training the target classifier. Since tampering the training process has no guarantees on the confidence score vectors, these defenses have no formal utility-loss guarantees on the confidence score vectors. Moreover, these defenses achieve suboptimal tradeoffs between the membership privacy of the training dataset and utility loss of the confidence score vectors. For instance, Jayaraman and Evans [25] found that existing differentially private machine learning methods rarely offer acceptable privacy-utility tradeoffs for complex models.

Our work: In this work, we propose *MemGuard*, the first defense with formal utility-loss guarantees against membership inference attacks under the black-box setting. Instead of tampering the training process of the target classifier, MemGuard randomly adds noise to the confidence score vector predicted by the target classifier for any query data sample. MemGuard can be applied to an existing target classifier without retraining it. Given a query data sample's confidence score vector, MemGuard aims to achieve two goals: 1) the attacker's classifier is inaccurate at inferring member or non-member for the query data sample after adding noise to the confidence score vector, and 2) the utility loss of the confidence score vector is bounded. Specifically, the noise should not change the predicted label of the query data sample, since even 1% loss of the label accuracy may be intolerable in some critical applications such as finance and healthcare. Moreover, the confidence score distortion introduced by the noise should be bounded by a budget since a confidence score vector intends to tell a user more information beyond the predicted label. We formulate achieving the two goals as solving an optimization problem. However, it is computationally challenging to solve the optimization problem as the noise space is large. To address the challenge, we propose a two-phase framework to approximately solve the problem.

We observe that an attacker uses an ML classifier to predict member or non-member and classifier can be misled by *adversarial examples* [10, 19, 31, 47–50, 62]. Therefore, in Phase I, MemGuard finds a carefully crafted noise vector that can turn the confidence score vector into an adversarial example. Specifically, MemGuard aims to find a noise vector such that the attacker's classifier is likely to make a random guessing at inferring member or non-member based on the noisy confidence score vector. Since the defender does not know the attacker's classifier as there are many choices, the defender itself trains a classifier for membership inference and crafts the noise vector based on its own classifier. Due to *transferability* [31, 32, 47, 62] of adversarial examples, the noise vector that misleads the defender's classifier is likely to also mislead the attacker's classifier. The adversarial machine learning community has developed many algorithms (e.g., [10, 19, 31, 35, 39, 40, 50, 63]) to find adversarial noise/examples. However, these algorithms are insufficient for our problem because they did not consider the unique constraints on utility loss of the confidence score vector. Specifically, the noisy confidence score vector should not change the predicted label of the query data sample and should still be a probability distribution. To address this challenge, we design a new algorithm to find a small noise vector that satisfies the utility-loss constraints.

In Phase II, MemGuard adds the noise vector found in Phase I to the true confidence score vector with a certain probability. The probability is selected such that the expected confidence score distortion is bounded by the budget and the defender's classifier is most likely to make random guessing at inferring member or non-member. Formally, we formulate finding this probability as solving an optimization problem and derive an analytical solution for the optimization problem.

We evaluate MemGuard and compare it with state-of-the-art defenses [1, 42, 56, 58] on three real-world datasets. Our empirical results show that MemGuard can effectively defend against state-of-the-art black-box membership inference attacks [43, 56]. In particular, as MemGuard is allowed to add larger noise (we measure the magnitude of the noise using its L_1-norm), the inference accuracies of all evaluated membership inference attacks become smaller. Moreover, MemGuard achieves better privacy-utility tradeoffs than state-of-the-art defenses. Specifically, given the same average confidence score distortion, MemGuard reduces the attacker's inference accuracy at inferring member/non-members by the most.

In summary, our key contributions are as follows:

- We propose MemGuard, the first defense with formal utility-loss guarantees against membership inference attacks under the black-box setting.
- We propose a new algorithm to find a noise vector that satisfies the unique utility-loss constraints in Phase I of MemGuard. Moreover, in Phase II, we derive an analytical solution of the probability with which MemGuard adds the noise vector to the confidence score vector.
- We evaluate MemGuard on three real-world datasets. Our results show that MemGuard is effective and outperforms existing defenses.

2 RELATED WORK

2.1 Membership Inference

Membership inference attacks: The goal of membership inference is to determine whether a certain data sample is inside a dataset. Homer et al. [23] proposed the first membership inference attack in the biomedical setting, in particular on genomic data. Specifically, they showed that an attacker can compare a user's genomic data with the summary statistics of the target database, such as mean and standard deviation, to determine the presence of the user in the database. The comparison can be done by using

statistical testing methods such as log-likelihood ratio test. Later, several works performed similar membership inference attacks against other types of biomedical data such as MicroRNA [4] and DNA methylation [20]. Recently, Pyrgelis et al. [52, 53] further showed that membership inference can also be performed effectively against location databases. In particular, they showed that an attacker can infer whether a user's location dataset was used for computing a given aggregate location dataset.

Membership inference attacks against ML models: Shokri et al. [58] introduced membership inference in the ML setting. The goal here is to determine whether a data sample is in the training dataset of a target black-box ML classifier. To achieve the goal, the attacker trains binary ML classifiers, which take a data sample's confidence score vector predicted by the target classifier as input and infer the data sample to be a member or non-member of the target classifier's training dataset. We call these classifiers *attack classifiers* and they are trained using *shadow classifiers*. Specifically, the attacker is assumed to have a dataset coming from the same distribution as the target classifier's training dataset and the attacker uses the dataset to train shadow classifiers, each of which aims to replicate the target classifier. Then, the attacker trains the attack classifiers by using the confidence score vectors predicted by the shadow classifiers for some members and non-members of the shadow classifiers' training datasets.

Salem et al. [56] recently proposed new membership inference attacks for black-box target classifiers, which relax the assumptions of the attacks proposed by Shokri et al. from both model and data angles. For instance, they showed that the attacker can rank the entries in a confidence score vector before feeding it into an attack classifier, which improves the attack effectiveness. Moreover, they showed that it is sufficient for the attacker to train just one shadow classifier. These results indicate that membership inference threat is even larger than previously thought.

More recently, Nasr et al. [43] proposed membership inference attacks against white-box ML models. For a data sample, they calculate the corresponding gradients over the white-box target classifier's parameters and use these gradients as the data sample's feature for membership inference. Moreover, both Nasr et al. [43] and Melis et al. [36] proposed membership inference attacks against federated learning. While most of the previous works concentrated on classification models [33, 34, 42, 43, 56, 58, 69], Hayes et al. [21] studied membership inference against generative models, in particular generative adversarial networks (GANs) [18]. They designed attacks for both white- and black-box settings. Their results showed that generative models are also vulnerable to membership inference.

Defense mechanisms against membership inference: Multiple defense mechanisms have been proposed to mitigate the threat of membership inference in the ML setting. We summarize them as the following.

L_2-Regularizer [58]. Overfitting, i.e., ML classifiers are more confident when facing data samples they are trained on (members) than others, is one major reason why membership inference is effective. Therefore, to defend against membership inference, people have explored to reduce overfitting using regularization. For instance, Shokri et al. [58] explored using conventional L_2 regularizer when training the target classifier.

Min-Max Game [42]. Nasr et al. [42] proposed a min-max game-theoretic method to train a target classifier. Specifically, the method formulates a min-max optimization problem that aims to minimize the target classifier's prediction loss while maximizing the membership privacy. This formulation is equivalent to adding a new regularization term called *adversarial regularization* to the loss function of the target classifier.

Dropout [56]. Dropout is a recently proposed technique to regularize neural networks [61]. Salem et al. [56] explored using dropout to mitigate membership inference attacks. Roughly speaking, dropout drops a neuron with a certain probability in each iteration of training a neural network.

Model Stacking [56]. Model stacking is a classical ensemble method which combines multiple weak classifiers' results as a strong one. Salem et al. [56] explored using model stacking to mitigate membership inference attacks. Specifically, the target classifier consists of three classifiers organized into a two-level tree structure. The first two classifiers on the bottom of the tree take the original data samples as input, while the third one's input is the outputs of the first two classifiers. The three classifiers are trained using disjoint sets of data samples, which reduces the chance for the target classifier to remember any specific data sample, thus preventing overfitting.

Differential privacy. Differential privacy [13] is a classical method for privacy-preserving machine learning. Most differential privacy based defenses add noise to the objective function that is used to learn a model [12, 24, 30], or the gradient in each iteration of gradient descent or stochastic gradient descent that is used to minimize the objective function [1, 6, 60, 66, 70]. Shokri and Shmatikov [57] designed a differential privacy method for collaborative learning of deep neural networks.

Limitations. Existing defenses suffer from two key limitations: 1) they do not have formal utility loss guarantee of the confidence score vector; and 2) they achieve suboptimal privacy-utility tradeoffs. Our defense addresses these two limitations. For instance, as we will show in experiments, with the same utility loss of the confidence score vector (e.g., the same L_1-norm distortion of the confidence score vector), our defense reduces the attack classifier's accuracy at inferring members/non-members to a larger extent than existing defenses.

Other privacy/confidentiality attacks against ML: There exist multiple other types of privacy/confidentiality attacks against ML models [2, 14–16, 36, 44, 55, 64, 65]. Fredrikson et al. [14, 15] proposed *model inversion attacks*. For instance, they can infer the missing values of an input feature vector by leveraging a classifier's prediction on the input feature vector. Several works [2, 16, 36] studied *property inference attacks*, which aim to infer a certain property (e.g., the fraction of male and female users) of a target classifier's training dataset. Tramèr et al. [64] proposed *model stealing attacks*. They designed different techniques tailored to different ML models aiming at stealing the parameters of the target models. Another line of works studied *hyperparameter stealing attacks* [44, 65], which aim to steal the hyperparameters such as the neural network architecture and the hyperparameter that balances between the loss function and the regularization term.

Table 1: Notations

Notation	Description
$\mathbf{x}$	A data sample
$\mathbf{s}$	A true confidence score vector
$\mathbf{s}'$	A noisy confidence score vector
$\mathbf{n}$	A noise vector
f	Decision function of the target classifier
$\mathbf{z}$	Logits of the target classifier
C	Attacker's attack classifier for membership inference
g	Decision function of defender's defense classifier
h	Logits of the defender's defense classifier
$\mathcal{M}$	Randomized noise addition mechanism
ϵ	Confidence score distortion budget

2.2 Adversarial Examples

Given a classifier and an example, we can add carefully crafted noise to the example such that the classifier predicts its label as we desire. The example with carefully crafted noise is called an *adversarial example*. Our MemGuard adds carefully crafted noise to a confidence score vector to turn it into an adversarial example, which is likely to mislead the attack classifier to make a random guessing at member or non-member. The adversarial machine learning community has developed many algorithms (e.g., [10, 19, 31, 35, 39, 40, 50, 63]) to find adversarial examples. However, these algorithms are insufficient to our problem because they did not consider the utility-loss constraints on the confidence score vectors. We address these challenges via designing a new algorithm to find adversarial examples.

Since our defense leverages adversarial examples to mislead the attacker's attack classifier, an adaptive attacker can leverage a classifier that is more robust against adversarial examples as the attack classifier. Although different methods (e.g., adversarial training [19, 35, 63], defensive distillation [51], Region-based Classification [9], MagNet [37], and Feature Squeezing [68]) have been explored to make classifiers robust against adversarial examples, it is still considered an open challenge to design such robust classifiers. Nevertheless, in our experiments, we will consider the attacker uses adversarial training to train its attack classifier, as adversarial training was considered to be the most empirically robust method against adversarial examples so far [3].

3 PROBLEM FORMULATION

In our problem formulation, we have three parties, i.e., *model provider*, *attacker*, and *defender*. Table 1 shows some important notations used in this paper.

3.1 Model Provider

We assume a model provider has a proprietary training dataset (e.g., healthcare dataset, location dataset). The model provider trains a machine learning classifier using the proprietary training dataset. Then, the model provider deploys the classifier as a cloud service or a client-side AI software product (e.g., a mobile or IoT app), so other users can leverage the classifier to make predictions for their own data samples. In particular, we consider the deployed classifier returns a confidence score vector for a query data sample. Formally, we have:

$$f : \mathbf{x} \mapsto \mathbf{s},$$

where f, $\mathbf{x}$, and $\mathbf{s}$ represent the classifier's decision function, the query data sample, and the confidence score vector, respectively. The confidence score vector essentially is the predicted posterior probability distribution of the label of the query data sample, i.e., s_j is the predicted posterior probability that the query data sample has label j. The label of the query data sample is predicted to be the one that has the largest confidence score, i.e., the label is predicted as $\text{argmax}_j\{s_j\}$. For convenience, we call the model provider's classifier *target classifier*. Moreover, we consider the target classifier is neural network in this work.

3.2 Attacker

An attacker aims to infer the proprietary training dataset of the model provider. Specifically, we consider the attacker only has *black-box* access to the target classifier, i.e., the attacker can send query data samples to the target classifier and obtain their confidence score vectors predicted by the target classifier. The attacker leverages black-box *membership inference attacks* [34, 42, 56, 58] to infer the members of the target classifier's training dataset. Roughly speaking, in membership inference attacks, the attacker trains a binary classifier, which takes a query data sample's confidence score vector as input and predicts whether the query data sample is in the target classifier's training dataset or not. Formally, we have:

$$C : \mathbf{s} \mapsto \{0, 1\},$$

where C is the attacker's binary classifier, $\mathbf{s}$ is the confidence score vector predicted by the target classifier for the query data sample $\mathbf{x}$, 0 indicates that the query data sample $\mathbf{x}$ is not a member of the target classifier's training dataset, and 1 indicates that the query data sample $\mathbf{x}$ is a member of the target classifier's training dataset. For convenience, we call the attacker's binary classifier C *attack classifier*. We will discuss more details about how the attacker could train its attack classifier in Section 5. Note that, to consider strong attacks, we assume the attacker knows our defense mechanism, but the defender does not know the attack classifier since the attacker has many choices for the attack classifier.

3.3 Defender

The defender aims to defend against black-box membership inference attacks. The defender could be the model provider itself or a trusted third party. For any query data sample from any user, the target classifier predicts its confidence score vector and the defender adds a *noise vector* to the confidence score vector before returning it to the user. Formally, we have:

$$\mathbf{s}' = \mathbf{s} + \mathbf{n},$$

where $\mathbf{s}$ is the true confidence score vector predicted by the target classifier for a query data sample, $\mathbf{n}$ is the noise vector added by the defender, and $\mathbf{s}'$ is the noisy confidence score vector that is returned to a user. Therefore, an attacker only has access to the noisy confidence score vectors. The defender aims to add noise to achieve the following two goals:

- **Goal I.** The attacker's attack classifier is inaccurate at inferring the members/non-members of the target classifier's

training dataset, i.e., protecting the privacy of the training dataset.

- **Goal II.** The utility loss of the confidence score vector is bounded.

However, achieving these two goals faces several challenges which we discuss next.

Achieving Goal I: The first challenge to achieve Goal I is that the defender does not know the attacker's attack classifier. To address the challenge, the defender itself trains a binary classifier to perform membership inference and adds noise vectors to the confidence score vectors such that its own classifier is inaccurate at inferring members/non-members. In particular, the defender's classifier takes a confidence score vector as input and predicts member or non-member for the corresponding data sample. We call the defender's binary classifier *defense classifier* and denote its decision function as g. Moreover, we consider the decision function $g(\mathbf{s})$ represents the probability that the corresponding data sample, whose confidence score vector predicted by the target classifier is $\mathbf{s}$, is a member of the target classifier's training dataset. In particular, we consider the defender trains a neural network classifier, whose output layer has one neuron with sigmoid activation function. For such classifier, the decision function's output (i.e., the output of the neuron in the output layer) represents probability of being a member. Formally, we have:

$$g : \mathbf{s} \mapsto [0, 1].$$

The defense classifier predicts a data sample to be member of the target classifier's training dataset if and only if $g(\mathbf{s}) > 0.5$.

To make the defense classifier inaccurate, one method is to add a noise vector to a true confidence score vector such that the defense classifier makes an incorrect prediction. Specifically, if the defense classifier predicts member (or non-member) for the true confidence score vector, then the defender adds a noise vector such that the defense classifier predicts non-member (or member) for the noisy confidence score vector. However, when an attacker knows the defense mechanism, the attacker can easily adapt its attack to achieve a high accuracy. In particular, the attacker predicts member (or non-member) when its attack classifier predicts non-member (or member) for a data sample. Another method is to add noise vectors such that the defense classifier always predicts member (or non-member) for the noisy confidence score vectors. However, for some true confidence score vectors, such method may need noise that violates the utility-loss constraints of the confidence score vectors (we will discuss utility-loss constraints later in this section).

Randomized noise addition mechanism. Therefore, we consider the defender adopts a *randomized noise addition mechanism* denoted as $\mathcal{M}$. Specifically, given a true confidence score vector $\mathbf{s}$, the defender samples a noise vector $\mathbf{n}$ from the space of possible noise vectors with a probability $\mathcal{M}(\mathbf{n}|\mathbf{s})$ and adds it to the true confidence score vector. Since random noise is added to a true confidence score vector, the decision function g outputs a random probability of being member. We consider the defender's goal is to make the expectation of the probability of being member predicted by g close to 0.5. In other words, the defender's goal is to add random noise such that the defense classifier randomly guesses member or non-member for a data sample on average. Formally, the defender aims to find a randomized noise addition mechanism $\mathcal{M}$ such that $|E_{\mathcal{M}}(g(\mathbf{s}+\mathbf{n})) - 0.5|$ is minimized.

Achieving Goal II: The key challenge to achieve Goal II is how to quantify the utility loss of the confidence score vector. To address the challenge, we introduce two utility-loss metrics.

Label loss. Our first metric concentrates on the query data sample's label predicted by the target classifier. Recall that the label of a query data sample is predicted as the one that has the largest confidence score. If the true confidence score vector and the noisy confidence score vector predict the same label for a query data sample, then the *label loss* is 0 for the query data sample, otherwise the label loss is 1 for the query data sample. The overall label loss of a defense mechanism is the label loss averaged over all query data samples. In some critical applications such as finance and healthcare, even 1% of label loss may be intolerable. In this work, we aim to achieve 0 label loss, i.e., our noise does not change the predicted label for any query data sample. Formally, we aim to achieve $\text{argmax}_j\{s_j\} = \text{argmax}_j\{s_j + n_j\}$, where $\text{argmax}_j\{s_j\}$ and $\text{argmax}_j\{s_j + n_j\}$ are the labels predicted based on the true and noisy confidence score vectors, respectively.

Confidence score distortion. The confidence score vector for a query data sample tells the user more information about the data sample's label beyond the predicted label. Therefore, the added noise should not substantially distort the confidence score vector. First, the noisy confidence score vector should still be a probability distribution. Formally, we have $s_j + n_j \geq 0$ for $\forall j$ and $\sum_j (s_j + n_j) = 1$. Second, the distance $d(\mathbf{s}, \mathbf{s}+\mathbf{n})$ between the true confidence score vector and the noisy confidence score vector should be small. In particular, we consider the model provider specifies a confidence score distortion budget called ϵ, which indicates the upper bound of the expected confidence score distortion that the model provider can tolerate. Formally, we aim to achieve $E_{\mathcal{M}}(d(\mathbf{s}, \mathbf{s}+\mathbf{n})) \leq \epsilon$. While any distance metric can be used to measure the distortion, we consider L_1-norm of the noise vector as the distance metric, i.e., $d(\mathbf{s}, \mathbf{s}+\mathbf{n}) = ||\mathbf{n}||_1$. We adopt L_1-norm of the noise vector because it is easy to interpret. Specifically, the L_1-norm of the noise vector is simply the sum of the absolute value of its entries.

Membership inference attack defense problem: After quantifying Goal I and Goal II, we can formally define our problem of defending against membership inference attacks.

Definition 3.1 (Membership-Inference-Attack Defense Problem). Given the decision function g of the defense classifier, a confidence score distortion budget ϵ, a true confidence score vector $\mathbf{s}$, the defender aims to find a randomized noise addition mechanism $\mathcal{M}^*$ via solving the following optimization problem:

$$\mathcal{M}^* = \underset{\mathcal{M}}{\text{argmin}} \; |E_{\mathcal{M}}(g(\mathbf{s}+\mathbf{n})) - 0.5| \tag{1}$$

$$\text{subject to: } \underset{j}{\text{argmax}}\{s_j + n_j\} = \underset{j}{\text{argmax}}\{s_j\} \tag{2}$$

$$E_{\mathcal{M}}(d(\mathbf{s}, \mathbf{s}+\mathbf{n})) \leq \epsilon \tag{3}$$

$$s_j + n_j \geq 0, \forall j \tag{4}$$

$$\sum_j s_j + n_j = 1, \tag{5}$$

where the objective function of the optimization problem is to achieve Goal I and the constraints are to achieve Goal II. Specifically,

the first constraint means that the added noise does not change the predicted label of the query data sample; the second constraint means that the confidence score distortion is bounded by the budget ϵ; and the last two constraints mean that the noisy confidence score vector is still a probability distribution. Note that the last constraint is equivalent to $\sum_j n_j = 0$ since $\sum_j s_j = 1$. Moreover, we adopt L_1-norm of the noise vector to measure the confidence score distortion, i.e., $d(\mathbf{s}, \mathbf{s}+\mathbf{n}) = ||\mathbf{n}||_1$.

4 OUR MemGuard

4.1 Overview

Finding the randomized noise addition mechanism is to solve the optimization problem in Equation 1. We consider two scenarios depending on whether $g(\mathbf{s})$ is 0.5 or not.

Scenario I: In this scenario, $g(\mathbf{s}) = 0.5$. For such scenario, it is easy to solve the optimization problem in Equation 1. Specifically, the mechanism that adds the noise vector $\mathbf{0}$ with probability 1 is the optimal randomized noise addition mechanism, with which the objective function has a value of 0.

Scenario II: In this scenario, $g(\mathbf{s})$ is not 0.5. The major challenge to solve the optimization problem in this scenario is that the randomized noise addition mechanism is a probability distribution over the continuous noise space for a given true confidence score vector. The noise space consists of the noise vectors that satisfy the four constraints of the optimization problem. As a result, it is challenging to represent the probability distribution and solve the optimization problem. To address the challenge, we observe that the noise space can be divided into two groups depending on the output of the defense classifier's decision function g. Specifically, for noise vectors in one group, if we add any of them to the true confidence score vector, then the decision function g outputs 0.5 as the probability of being member. For noise vectors in the other group, if we add any of them to the true confidence score vector, then the decision function g outputs a probability of being member that is not 0.5.

Based on this observation, we propose a *two-phase framework* to approximately solve the optimization problem. Specifically, in Phase I, for each noise group, we find the noise vector with minimum confidence score distortion (i.e., $d(\mathbf{s}, \mathbf{s}+\mathbf{n})$ is minimized) as a *representative* noise vector for the noise group. We select the noise vector with minimum confidence score distortion in order to minimize the confidence score distortion. Since $g(\mathbf{s}) \neq 0.5$, the selected representative noise vector for the second noise group is $\mathbf{0}$. We denote by $\mathbf{r}$ the selected representative noise vector for the first noise group. In Phase II, we assume the randomized noise addition mechanism is a probability distribution over the two representative noise vectors instead of the overall noise space. Specifically, the defender adds the representative noise vector $\mathbf{r}$ to the true confidence score vector with a certain probability and does not add any noise with the remaining probability.

Next, we introduce our Phase I and Phase II.

4.2 Phase I: Finding r

Finding r as solving an optimization problem: Our goal essentially is to find a noise vector $\mathbf{r}$ such that 1) the utility loss of the confidence score vector is minimized and 2) the decision function g outputs 0.5 as the probability of being member when taking the noisy confidence score vector as an input. Formally, we find such noise vector via solving the following optimization problem:

$$\min_{\mathbf{r}} d(\mathbf{s}, \mathbf{s}+\mathbf{r}) \tag{6}$$

$$\text{subject to: } \operatorname*{argmax}_j\{s_j + r_j\} = \operatorname*{argmax}_j\{s_j\} \tag{7}$$

$$g(\mathbf{s}+\mathbf{r}) = 0.5 \tag{8}$$

$$s_j + r_j \geq 0, \forall j \tag{9}$$

$$\sum_j r_j = 0, \tag{10}$$

where $\mathbf{s}$ is the true confidence score vector, the objective function means that the confidence score distortion is minimized, the first constraint means that the noise does not change the predicted label of the query data sample, the second constraint means that the defense classifier's decision function outputs 0.5 (i.e., the defense classifier's prediction is random guessing), and the last two constraints mean that the noisy confidence score vector is still a probability distribution.

Solving the optimization problem in Equation 6 can be viewed as finding an *adversarial example* to evade the defense classifier. In particular, $\mathbf{s}$ is a normal example and $\mathbf{s}+\mathbf{r}$ is an adversarial example. The adversarial machine learning community has developed many algorithms (e.g., [10, 19, 31, 35, 39, 40, 50, 63]) to find adversarial examples. However, these algorithms are insufficient to our problem because they did not consider the unique challenges of privacy protection. In particular, they did not consider the utility-loss constraints, i.e., the constraints in Equation 7, Equation 9, and Equation 10.

One naive method (we call it *Random*) to address the challenges is to generate a random noise vector that satisfies the utility-loss constraints. In particular, we can generate a random vector $\mathbf{r}'$ whose entries are non-negative and sum to 1. For instance, we first sample a number r'_1 from the interval [0,1] uniformly at random as the first entry. Then, we sample a number r'_2 from the interval [0, 1-r'_1] uniformly at random as the second entry. We repeat this process until the last entry is 1 minus the sum of the previous entries. Then, we exchange the largest entry of $\mathbf{r}'$ to the position j to satisfy the constraint 7. Finally, we treat $\mathbf{r} = \mathbf{r}' - \mathbf{s}$ as the noise vector, which is a solution to the optimization problem in Equation 6. However, as we will show in experiments, this Random method achieves suboptimal privacy-utility tradeoffs because the noise vector is not optimized and it is challenging to satisfy the constraint Equation 9. We propose to solve the optimization problem via change of variables and adding the constraints to the objective function.

Eliminating the constraints on probability distribution via change of variables: Since we consider the target classifier to be a neural network, whose output layer is a softmax layer, the true confidence score vector $\mathbf{s}$ is a softmax function of some vector $\mathbf{z}$. The vector $\mathbf{z}$ is the output of the neurons in the second-to-last layer of the neural network and is often called *logits* of the neural network. Formally, we have:

$$\mathbf{s} = softmax(\mathbf{z}). \tag{11}$$

Moreover, we model the noisy confidence score vector as follows:

$$\mathbf{s} + \mathbf{r} = softmax(\mathbf{z} + \mathbf{e}), \tag{12}$$

where $\mathbf{e}$ is a new vector variable. For any value of $\mathbf{e}$, the noisy confidence score vector $\mathbf{s} + \mathbf{r}$ is a probability distribution, i.e., the constraints in Equation 9 and Equation 10 are satisfied. Therefore, in the optimization problem in Equation 6, we change the true confidence score vector $\mathbf{s}$ as $softmax(\mathbf{z})$ and change the variable $\mathbf{r}$ as $softmax(\mathbf{z} + \mathbf{e}) - softmax(\mathbf{z})$. Then, we obtain the following optimization problem:

$$\min_{\mathbf{e}} d(softmax(\mathbf{z}), softmax(\mathbf{z} + \mathbf{e})) \tag{13}$$

$$\text{subject to: } \operatorname*{argmax}_j\{z_j + e_j\} = \operatorname*{argmax}_j\{z_j\} \tag{14}$$

$$g(softmax(\mathbf{z} + \mathbf{e})) = 0.5. \tag{15}$$

After solving $\mathbf{e}$ in the above optimization problem, we can obtain the noise vector $\mathbf{r}$ as follows:

$$\mathbf{r} = softmax(\mathbf{z} + \mathbf{e}) - softmax(\mathbf{z}). \tag{16}$$

The optimization problem without the constraints on probability distribution is still challenging to solve because the remaining two constraints are highly nonlinear. To address the challenge, we turn the constraints into the objective function.

Turning the constraint in Equation 15 into the objective function: We consider the defender's binary defense classifier is a neural network whose output layer has a single neuron with sigmoid activation function. Therefore, we have:

$$g(softmax(\mathbf{z} + \mathbf{e})) = \frac{1}{1 + \exp(-h(softmax(\mathbf{z} + \mathbf{e})))}, \tag{17}$$

where $h(softmax(\mathbf{z} + \mathbf{e}))$ is the output of the neuron in the second-to-last layer of the defense classifier when the defense classifier takes the noisy confidence score vector $softmax(\mathbf{z} + \mathbf{e})$ as an input. In other words, h is the logit of the defense classifier. $g(softmax(\mathbf{z} + \mathbf{e})) = 0.5$ implies $h(softmax(\mathbf{z} + \mathbf{e})) = 0$. Therefore, we transform the constraint in Equation 15 to the following loss function:

$$L_1 = |h(softmax(\mathbf{z} + \mathbf{e}))|, \tag{18}$$

where L_1 is small when $h(softmax(\mathbf{z} + \mathbf{e}))$ is close to 0.

Turning the constraint in Equation 14 into the objective function: We denote by l the predicted label for the query data sample, i.e., $l = argmax_j\{s_j\} = \operatorname{argmax}_j\{z_j\}$. The constraint in Equation 14 means that $z_l + e_l$ is the largest entry in the vector $\mathbf{z} + \mathbf{e}$. Therefore, we enforce the inequality constraint $z_l + e_l \geq max_{j|j\neq l}\{z_j + e_j\}$. Moreover, we further transform the inequality constraint to the following loss function:

$$L_2 = \text{ReLU}(-z_l - e_l + max_{j|j\neq l}\{z_j + e_j\}), \tag{19}$$

where the function ReLU is defined as $\text{ReLU}(v)=\max\{0, v\}$. The loss function L_2 is 0 if the inequality $z_l + e_l \geq max_{j|j\neq l}\{z_j + e_j\}$ holds.

Unconstrained optimization problem: After transforming the constraints into the objective function, we have the following unconstrained optimization problem:

$$\min_{\mathbf{e}} L = L_1 + c_2 \cdot L_2 + c_3 \cdot L_3, \tag{20}$$

Algorithm 1 Phase I of MemGuard

Input: $\mathbf{z}$, max_iter, c_2, c_3, and β (learning rate).
Output: $\mathbf{e}$

1: //Predicted label
2: $l = \operatorname{argmax}_j\{z_j\}$
3: **while** $True$ **do**
4: //A new iteration to search c_3
5: $\mathbf{e} = \mathbf{0}$
6: $\mathbf{e}' = \mathbf{e}$
7: $i = 1$
8: **while** $i < max\_iter$ and ($\operatorname{argmax}_j\{z_j + e_j\} \neq l$ or $h(softmax(\mathbf{z})) \cdot h(softmax(\mathbf{z} + \mathbf{e})) > 0$) **do**
9: //Gradient descent with normalized gradient
10: $\mathbf{u} = \frac{\partial L}{\partial \mathbf{e}}$
11: $\mathbf{u} = \mathbf{u}/||\mathbf{u}||_2$
12: $\mathbf{e} = \mathbf{e} - \beta \cdot \mathbf{u}$
13: $i = i + 1$
14: **end while**
15: //Return the vector in the previous iteration if the predicted label changes or the sign of h does not change in the current iteration
16: **if** $\operatorname{argmax}_j\{z_j + e_j\} \neq l$ or $h(softmax(\mathbf{z})) \cdot h(softmax(\mathbf{z} + \mathbf{e})) > 0$ **then**
17: **return** $\mathbf{e}'$
18: **end if**
19: $c_3 = 10 \cdot c_3$
20: **end while**

where $L_3 = d(softmax(\mathbf{z}), softmax(\mathbf{z} + \mathbf{e}))$, while c_2 and c_3 balance between the three terms.

Solving the unconstrained optimization problem: We design an algorithm based on gradient descent to solve the unconstrained optimization problem. Algorithm 1 shows our algorithm. Since we aim to find a noise vector that has a small confidence score distortion, we iteratively search a large c_3. For each given c_3, we use gradient descent to find $\mathbf{e}$ that satisfies the constraints in Equation 14 and Equation 15. The process of searching c_3 stops when we cannot find a vector $\mathbf{e}$ that satisfies the two constraints. Specifically, given c_2, c_3, and a learning rate β, we iteratively update the vector variable $\mathbf{e}$ (i.e., the inner while loop in Algorithm 1). Since we transform the constraints in Equation 14 and Equation 15 into the objective function, there is no guarantee that they are satisfied during the iterative gradient descent process. Therefore, in each iteration of gradient descent, we check whether the two constraints are satisfied (i.e., Line 8 in Algorithm 1). Specifically, we continue the gradient descent process when the predicted label changes or the sign of the logit h does not change. In other words, we stop the gradient descent process when both constraints are satisfied. We use $h(softmax(\mathbf{z})) \cdot h(softmax(\mathbf{z} + \mathbf{e})) \leq 0$ to approximate the constraint in Equation 15. In particular, the constraint in Equation 15 is equivalent to $h(softmax(\mathbf{z} + \mathbf{e})) = 0$. Once we find a vector $\mathbf{e}$ such that $h(softmax(\mathbf{z}))$ and $h(softmax(\mathbf{z} + \mathbf{e}))$ have different signs (e.g., $h(softmax(\mathbf{z})) > 0$ and $h(softmax(\mathbf{z} + \mathbf{e})) < 0$), $h(softmax(\mathbf{z} + \mathbf{e}))$

just crosses 0 and should be close to 0 since we use a small learning rate. Note that we could also iteratively search c_2, but it is computationally inefficient to search both c_2 and c_3.

4.3 Phase II

After Phase I, we have two representative noise vectors. One is $\mathbf{0}$ and the other is $\mathbf{r}$. In Phase II, we assume the randomized noise addition mechanism is a probability distribution over the two representative noise vectors instead of the entire noise space. Specifically, we assume that the defender picks the representative noise vectors $\mathbf{r}$ and $\mathbf{0}$ with probabilities p and $1-p$, respectively; and the defender adds the picked representative noise vector to the true confidence score vector. With such simplification, we can simplify the optimization problem in Equation 1 to the following optimization problem:

$$p = \underset{p}{\operatorname{argmin}} |p \cdot g(\mathbf{s}+\mathbf{r}) + (1-p) \cdot g(\mathbf{s}+\mathbf{0}) - 0.5| \tag{21}$$

$$\text{subject to: } p \cdot d(\mathbf{s}, \mathbf{s}+\mathbf{r}) + (1-p) \cdot d(\mathbf{s}, \mathbf{s}+\mathbf{0}) \leq \epsilon, \tag{22}$$

where the constraint means that the expected confidence score distortion is bounded by the budget. Note that we omit the other three constraints in Equation 2, Equation 4, and Equation 5. This is because both of our representative noise vectors already satisfy those constraints. Moreover, we can derive an analytical solution to the simplified optimization problem. The analytical solution is as follows:

$$p = \begin{cases} 0, & \text{if } |g(\mathbf{s}) - 0.5| \leq |g(\mathbf{s}+\mathbf{r}) - 0.5| \\ \min(\frac{\epsilon}{d(\mathbf{s}, \mathbf{s}+\mathbf{r})}, 1.0), & \text{otherwise.} \end{cases} \tag{23}$$

One-time randomness: If the defender randomly samples one of the two representative noise vectors every time for the same query data sample, then an attacker could infer the true confidence score vector via querying the same data sample multiple times. We consider the attacker knows our defense mechanism including the confidence score distortion metric d, the budget ϵ, and that the noise vector is sampled from two representative noise vectors, one of which is $\mathbf{0}$.

Suppose the attacker queries the same data sample n times from the target classifier. The attacker receives a confidence score vector $\mathbf{s}_1$ for m times and a confidence score vector $\mathbf{s}_2$ for $n-m$ times. One confidence score vector is $\mathbf{s}+\mathbf{r}$ and the other is the true confidence score vector $\mathbf{s}$. Since the attacker receives two different confidence score vectors, the attacker knows $0 < p < 1$. Moreover, given the two confidence score vectors, the attacker can compute p according to Equation 23 since the distance $d(\mathbf{s}, \mathbf{s}+\mathbf{r})$ does not depend on the ordering of $\mathbf{s}$ and $\mathbf{s}+\mathbf{r}$, i.e., $d(\mathbf{s}, \mathbf{s}+\mathbf{r}) = d(\mathbf{s}_1, \mathbf{s}_2)$. The attacker can also estimate the probabilities that the defender returns the confidence score vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ as $\frac{m}{n}$ and $\frac{n-m}{n}$, respectively. If $\frac{m}{n}$ is closer to p, then the attacker predicts that $\mathbf{s}_2$ is the true confidence score vector, otherwise the attacker predicts $\mathbf{s}_1$ to be the true confidence score vector.

To address this challenge, we propose to use one-time randomness when the defender samples the representative noise, with which the defender always returns the same confidence score vector for the same query data sample. Specifically, for a query data sample, the defender quantizes each dimension of the query data sample and computes the hash value of the quantized data sample. Then, the defender generates a random number p' in the range $[0, 1]$ via a pseudo random number generator with the hash value as the seed. If $p' < p$, the defender adds the representative noise vector $\mathbf{r}$ to the true confidence score vector, otherwise the defender does not add noise. The random number p' is the same for the same query data sample, so the defender always returns the same confidence score vector for the same query data sample. We compute the hash value of the quantized query data sample as the seed such that the attacker cannot just slightly modify the query data sample to generate a different p'. The attacker can compute the random number p' as we assume the attacker knows the defense mechanism including the hash function and pseudo random number generator. However, the attacker does not know p any more because the defender always returns the same confidence score vector for the same query data sample. Therefore, the attacker does not know whether the returned confidence score vector is the true one or not.

5 EVALUATION

5.1 Experimental Setup

5.1.1 Datasets. We use three datasets that represent different application scenarios.

Location: This dataset was preprocessed from the Foursquare dataset[1] and we obtained it from [58]. The dataset has 5,010 data samples with 446 binary features, each of which represents whether a user visited a particular region or location type. The data samples are grouped into 30 clusters. This dataset represents a 30-class classification problem, where each cluster is a class.

Texas100: This dataset is based on the Discharge Data public use files published by the Texas Department of State Health Services.[2] We obtained the preprocessed dataset from [58]. The dataset has $67,330$ data samples with $6,170$ binary features. These features represent the external causes of injury (e.g., suicide, drug misuse), the diagnosis, the procedures the patient underwent, and some generic information (e.g., gender, age, and race). Similar to [58], we focus on the 100 most frequent procedures and the classification task is to predict a procedure for a patient using the patient's data. This dataset represents a 100-class classification problem.

CH-MNIST: This dataset is used for classification of different tissue types on histology tile from patients with colorectal cancer. The dataset contains $5,000$ images from 8 tissues. The classification task is to predict tissue for an image, i.e., the dataset is a 8-class classification problem. The size of each image is 64×64. We obtained a preprocessed version from Kaggle. [3].

Dataset splits: For each dataset, we will train a target classifier, an attack classifier, and a defense classifier. Therefore, we split each dataset into multiple folds. Specifically, for the Location (or CH-MNIST) dataset, we randomly sample 4 disjoint sets, each of which includes 1,000 data samples. We denote them as D_1, D_2, D_3, and D_4, respectively. For the Texas100 dataset, we also randomly sample such 4 disjoint sets, but each set includes 10,000 data samples as the Texas100 dataset is around one order of magnitude larger. Roughly speaking, for each dataset, we use D_1, D_2, and D_3 to learn

[1] https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[2] https://www.dshs.texas.gov/THCIC/Hospitals/Download.shtm
[3] https://www.kaggle.com/kmader/colorectal-histology-mnist

Table 2: Neural network architecture of the target classifier for CH-MNIST.

Layer Type	Layer Parameters
Input 64×64	
Convolution Activation	$32 \times 3 \times 3$, strides=(1, 1), padding=same ReLU
Convolution Activation Pooling	$32 \times 3 \times 3$, strides=(1, 1) ReLU MaxPooling(2×2)
Convolution Activation	$32 \times 3 \times 3$, strides=(1, 1), padding=same ReLU
Convolution Activation	$32 \times 3 \times 3$, strides=(1, 1) ReLU
Pooling	MaxPooling(2×2)
Flatten	
Fully Connected	512
Fully Connected	8
Activation	softmax
Output	

the target classifier, the attack classifier, and the defense classifier, respectively; and we use $D_1 \cup D_4$ to evaluate the accuracy of the attack classifier. We will describe more details on how the sets are used when we use them.

5.1.2 Target Classifiers. For the Location and Texas100 datasets, we use a fully-connected neural network with 4 hidden layers as the target classifier. The number of neurons for the four layers are 1024, 512, 256, and 128, respectively. We use the popular activation function ReLU for the neurons in the hidden layers. The activation function in the output layer is softmax. We adopt the cross-entropy loss function and use Stochastic Gradient Descent (SGD) to learn the model parameters. We train 200 epochs with a learning rate 0.01, and we decay the learning rate by 0.1 in the 150th epoch for better convergence. For the CH-MNIST dataset, the neural network architecture of the target classifier is shown in Table 2. Similarly, we also adopt the cross-entropy loss function and use SGD to learn the model parameters. We train 400 epochs with a learning rate 0.01 and decay the learning rate by 0.1 in the 350th epoch. For each dataset, we use D_1 to train the target classifier. Table 3 shows the training and testing accuracies of the target classifiers on the three datasets, where the testing accuracy is calculated by using the target classifier to make predictions for the data samples that are not in D_1.

5.1.3 Membership Inference Attacks. In a membership inference attack, an attacker trains an attack classifier, which predicts *member* or *non-member* for a query data sample. The effectiveness of an attack is measured by the *inference accuracy* of the attack classifier, where the inference accuracy is the fraction of data samples in $D_1 \cup D_4$ that the attack classifier can correctly predict as member or non-member. In particular, data samples in D_1 are members of the target classifier's training dataset, while data samples in D_4 are non-members. We call the dataset $D_1 \cup D_4$ *evaluation dataset*. We consider two categories of state-of-the-art black-box membership inference attacks, i.e., *non-adaptive attacks* and *adaptive attacks*. In non-adaptive attacks, the attacker does not adapt its attack classifier based on our defense, while the attacker adapts its attack classifier based on our defense in adaptive attacks.

Table 3: Training and testing accuracies of the target classifier on the three datasets.

	Location	Texas100	CH-MNIST
Training Accuracy	100.0%	99.98%	99.0%
Testing Accuracy	60.32%	51.59%	72.0%

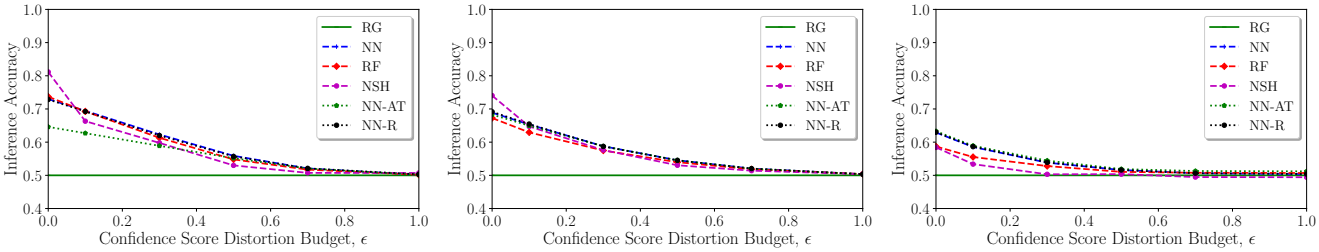
Non-adaptive attacks: We consider the *random guessing* attack and state-of-the-art attacks as follows.

Random guessing (RG) attack. For any query data sample, this attack predicts it to be a member of the target classifier's training dataset with probability 0.5. The inference accuracy of the RG attack is 0.5.

Neural Network (NN) attack [56, 58]. This attack assumes that the attacker knows the distribution of the target classifier's training dataset and the architecture of the target classifier. We further split the dataset D_2 into two halves denoted as D_2' and D_2'', respectively. The attacker uses D_2' to train a shadow classifier that has the same neural network architecture as the target classifier. After training the shadow classifier, the attacker calculates the confidence score vectors for the data samples in D_2' and D_2'', which are members and non-members of the shadow classifier. Then, the attacker ranks each confidence score vector and treats the ranked confidence score vectors of members and non-members as a "training dataset" to train an attack classifier. The attack classifier takes a data sample's ranked confidence score vector as an input and predicts member or non-member. For all three datasets, we consider the attack classifier is a fully-connected neural network with three hidden layers, which have 512, 256, and 128 neurons, respectively. The output layer just has one neuron. The neurons in the hidden layers use the ReLU activation function, while the neuron in the output layer uses the sigmoid activation function. The attack classifier predicts member if and only if the neuron in the output layer outputs a value that is larger than 0.5. We train the attack classifier for 400 epochs with a learning rate 0.01 using SGD and decay the learning rate by 0.1 at the 300th epoch.

Random Forest (RF) attack. This attack is the same as the NN attack except that RF attack uses random forest as the attack classifier, while NN uses a neural network as the attack classifier. We use scikit-learn with the default setting to learn random forest classifiers. We consider this RF attack to demonstrate that our defense mechanism is still effective even if the attack classifier and the defense classifier (a neural network) use different types of algorithms, i.e., the noise vector that evades the defense classifier can also evade the attack classifier even if the two classifiers use different types of algorithms.

NSH attack [42]. Nasr, Shokri, and Houmansadr [42] proposed this attack, which we abbreviate as NSH. This attack uses multiple neural networks. One network operates on the confidence score vector. Another one operates on the label which is one hot encoded. Both networks are fully-connected and have the same number of input dimension, i.e., the number of classes of the target classifier. Specifically, NSH assumes the attacker knows some members and



(a) Location **(b) Texas100** **(c) CH-MNIST**

Figure 1: Inference accuracies of different attacks as the confidence score distortion budget (i.e., ϵ) increases.

non-members of the target classifier's training dataset. In our experiments, we assume the attacker knows 30% of data samples in D_1 (i.e., members) and 30% of data samples in D_4 (i.e., non-members). The attacker uses these data samples to train the attack classifier. We adopt the neural network architecture in [42] as the attack classifier. The remaining 70% of data samples in D_1 and D_4 are used to calculate the inference accuracy of the attack classifier. We train the attack classifier for 400 epochs with an initial learning rate 0.01 and decay the learning rate by 0.1 after 300 epochs.

Adaptive attacks: We consider two attacks that are customized to our defense.

Adversarial training (NN-AT). One adaptive attack is to train the attack classifier via adversarial training, which was considered to be the most empirically robust method against adversarial examples so far [3]. We adapt the NN attack using adversarial training and denote the adapted attack as NN-AT. Specifically, for each data sample in D_2' and D_2'', the attacker calculates its confidence score vector using the shadow classifier. Then, the attacker uses the Phase I of our defense to find the representative noise vector and adds it to the confidence score vector to obtain a noisy confidence score vector. Finally, the attacker trains the attack classifier via treating the true confidence score vectors and their corresponding noisy versions of data samples in D_2' and D_2'' as a training dataset.

Rounding (NN-R). Since our defense adds carefully crafted small noise to the confidence score vector, an adaptive attack is to *round* each confidence score before using the attack classifier to predict member/non-member. Specifically, we consider the attacker rounds each confidence score to be one decimal and uses the NN attack. Note that rounding is also applied when training the NN attack classifier. We denote this attack NN-R.

Table 4 shows the inference accuracies of different attacks when our defense is not used. All attacks except RG have inference accuracies that are larger or substantially larger than 0.5.

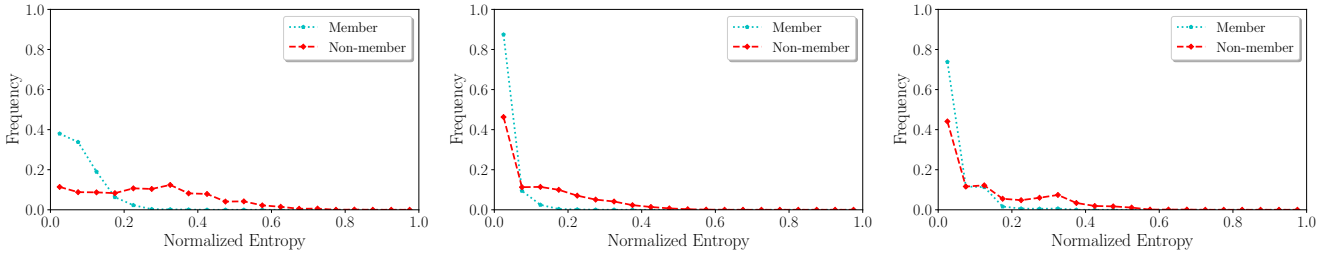
Table 4: Inference accuracies of different attacks on the three datasets when our defense is not used.

	Location	Texas100	CH-MNIST
RG	50.0%	50.0%	50.0%
NN	73.0%	68.9%	62.9%
RF	73.7%	67.3%	58.7%
NSH	81.1%	74.0%	58.4%
NN-AT	64.6%	68.3%	63.3%
NN-R	72.9%	69.2%	63.0%

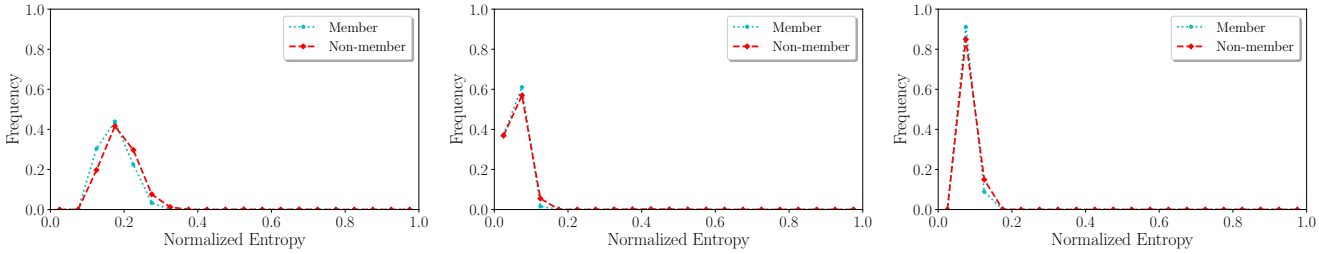
5.1.4 Defense Setting. In our defense, we need to specify a defense classifier and the parameters in Algorithm 1.

Defense classifier: The defender itself trains a classifier to perform membership inference. We consider the defense classifier is a neural network. However, since the defender does not know the attacker's attack classifier, we assume the defense classifier and the attack classifier use different neural network architectures. Specifically, we consider three different defense classifiers in order to study the impact of defense classifier on MemGuard. The three defense classifiers are fully-connected neural networks with 2, 3, and 4 hidden layers, respectively. The hidden layers of the three defense classifiers have (256, 128), (256, 128, 64), and (512, 256, 128, 64) neurons, respectively. The output layer has just one neuron. The activation function for the neurons in the hidden layers is *ReLU*, while the neuron in the output layer uses the sigmoid activation function. Unless otherwise mentioned, we use the defense classifier with 3 hidden layers. The defender calculates the confidence score vector for each data sample in D_1 and D_3 using the target classifier. The confidence score vectors for data samples in D_1 and D_3 have labels "member" and "non-member", respectively. The defender treats these confidence score vectors as a training dataset to learn a defense classifier, which takes a confidence score vector as an input and predicts member or non-member. We train a defense classifier for 400 epochs with a learning rate 0.001. We note that we can also synthesize data samples based on D_1 as non-members (Appendix A shows details).

Parameter setting: We set $max\_iter = 300$ and $\beta = 0.1$ in Algorithm 1. We found that once max_iter is larger than some threshold, MemGuard's effectiveness does not change. Since we aim to find representative noise vector that does not change the predicted label, we assign a relatively large value to c_2, which means that the objective function has a large value if the predicted label changes (i.e., the loss function L_2 is non-zero). In particular, we set $c_2 = 10$. Our Algorithm 1 searches for a large c_3 and we set the initial value of c_3 to be 0.1. We also compare searching c_2 with searching c_3.



(a) Location, without defense (b) Texas100, without defense (c) CH-MNIST, without defense



(d) Location, with defense (e) Texas100, with defense (f) CH-MNIST, with defense

Figure 2: Distribution of the normalized entropy of the confidence score vectors for members and non-members of the target classifier. Figures on the upper side are results without our defense, and figures on the lower side are results with our defense.

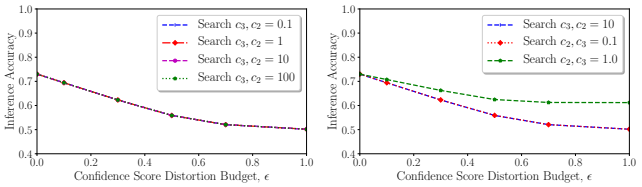
5.2 Experimental Results

MemGuard is effective: Figure 1 shows the inference accuracies of different attacks as the confidence score distortion budget increases on the three datasets. Since we adopt the expected L_1-norm of the noise vector to measure the confidence score distortion, the confidence score distortion is in the range [0, 2]. Note that our defense is guaranteed to achieve 0 label loss as our Algorithm 1 guarantees that the predicted label does not change when searching for the representative noise vector. We observe that our MemGuard can effectively defend against membership inference attacks, i.e., the inference accuracies of all the evaluated attacks decrease as our defense is allowed to add larger noise to the confidence score vectors. For instance, on Location, when our defense is allowed to add noise whose expected L_1-norm is around 0.8, our defense can reduce all the evaluated attacks to the random guessing (RG) attack; on CH-MNIST, our defense can reduce the NSH attack (or the remaining attacks) to random guessing when allowed to add noise whose expected L_1-norm is around 0.3 (or 0.7).

Indistinguishability between the confidence score vectors of members and non-members: We follow previous work [42] to study the distribution of confidence score vectors of members vs. non-members of the target classifier. Specifically, given a confidence score vector $\mathbf{s}$, we compute its *normalized entropy* as follows:

$$\textbf{Normalized entropy: } -\frac{1}{\log k}\sum_j s_j \log(s_j), \tag{24}$$

where k is the number of classes in the target classifier. Figure 2 shows the distributions of the normalized entropy of the confidence score vectors for members (i.e., data samples in D_1) and non-members (i.e., data samples in D_4) of the target classifier, where



(a) Searching c_3 (b) Searching c_2

Figure 3: Inference accuracy of the NN attack as the confidence score distortion budget increases on the Location dataset when searching c_3 or c_2.

we set the confidence score distortion budget ϵ to be 1 when our defense is used. The gap between the two curves in a graph corresponds to the information leakage of the target classifier's training dataset. Our defense substantially reduces such gaps. Specifically, the *maximum gap* between the two curves (without defense vs. with defense) is (0.27 vs. 0.11), (0.41 vs. 0.05), and (0.30 vs. 0.06) on the Location, Texas100, and CH-MNIST datasets, respectively. Moreover, the *average gap* between the two curves (without defense vs. with defense) is (0.062 vs. 0.011), (0.041 vs. 0.005), and (0.030 vs. 0.006) on the three datasets, respectively.

Searching c_2 vs. searching c_3: Figure 3a shows the inference accuracy of the NN attack as the confidence score distortion budget increases when fixing c_2 to different values and searching c_3. Figure 3b shows the results when fixing c_3 and searching c_2. We observe that MemGuard is insensitive to the setting of c_2 when searching c_3. Specifically, MemGuard has almost the same effectiveness when fixing c_2 to different values, i.e., the different curves

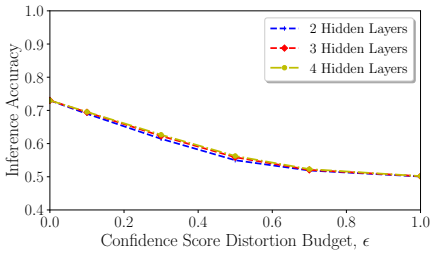


Figure 4: Inference accuracy of the NN attack as the confidence score distortion budget increases on the Location dataset when using different defense classifiers.

overlap in Figure 3a. This is because when our Phase I stops searching the noise vector, the predicted label is preserved, which means that the loss function L_2 is 0. However, MemGuard is sensitive to the setting of c_3 when searching c_2. Specifically, when fixing c_3 to be 0.1, searching c_2 achieves the same effectiveness as searching c_3. However, when fixing c_3 to be 1.0, searching c_2 is less effective. Therefore, we decided to search c_3 while fixing c_2.

Impact of defense classifiers: Figure 4 shows the inference accuracy of the NN attack as the confidence score distortion budget increases on the Location dataset when using different defense classifiers. We observe that MemGuard has similar effectiveness for different defense classifiers, which means that our carefully crafted noise vectors can transfer between classifiers.

MemGuard outperforms existing defenses: We compare with state-of-the-art defenses including L_2-Regularizer [58], Min-Max Game [42], Dropout [56], Model Stacking [56], and DP-SGD [1]. Each compared defense (except Model Stacking) has a hyperparameter to control the privacy-utility tradeoff. For instance, the hyperparameter that balances between the loss function and the L_2 regularizer in L_2-Regularizer, the hyperparameter that balances between the loss function and the adversarial regularizer in Min-Max Game, the dropout rate in Dropout, the privacy budget in DP-SGD, and ϵ in MemGuard. We also compare with MemGuard-Random in which we use the *Random* method (refer to Section 4.2) to generate the noise vector in Phase I.

Before deploying any defense, we use the undefended target classifier to compute the confidence score vector for each data sample in the evaluation dataset $D_1 \cup D_4$. For each defense and a given hyperparameter, we apply the defense to the target classifier and use the defended target classifier to compute the confidence score vector for each data sample in $D_1 \cup D_4$. Then, we compute the confidence score distortion for each data sample and obtain the *average confidence score distortion* on the evaluation dataset $D_1 \cup D_4$. Moreover, we compute the inference accuracy of the attack classifier (we consider NN in these experiments) on the evaluation dataset after the defense is used. Therefore, for each defense and a given hyperparameter, we can obtain a pair (inference accuracy, average confidence score distortion). Via exploring different hyperparameters, we can obtain a set of such pairs for each defense. Then, we plot these pairs on a graph, which is shown in Figure 5.

Table 5: Results of Model Stacking.

	Location	Texas100	CH-MNIST
Inference Acc.	50.0%	50.8%	50.0%
Average Distortion	1.63	1.28	0.81
Label Loss	56.3%	37.9%	18.3%

Specifically, we tried the hyperparameter of L_2-Regularizer in the range $[0, 0.05]$ with a step size 0.005, 0.001, and 0.005 for Location, Texas100, and CH_MNIST datasets, respectively. We tried the hyperparameter of Min-Max Game in the range $[0, 3]$ with a step size 0.5. We tried the dropout rate of Dropout in the range $[0, 0.9]$ with a step size 0.1. We use a publicly available implementation[4] of DP-SGD. We tried the parameter *noise_multiplier* that controls the privacy budget in the range $[0, 0.2]$ with a step size 0.05. We tried $[0, 0.1, 0.3, 0.5, 0.7, 1.0]$ as the ϵ in MemGuard and MemGuard-Random.

Our results show that MemGuard achieves the best privacy-utility tradeoff. In particular, given the same average confidence score distortion, MemGuard achieves the smallest inference accuracy. According to the authors of Model Stacking, it does not have a hyperparameter to easily control the privacy-utility tradeoff. Therefore, we just obtain one pair of (inference accuracy, average confidence score distortion) and Table 5 shows the results. Model Stacking reduces the inference accuracy to be close to 0.5, but the utility loss is intolerable.

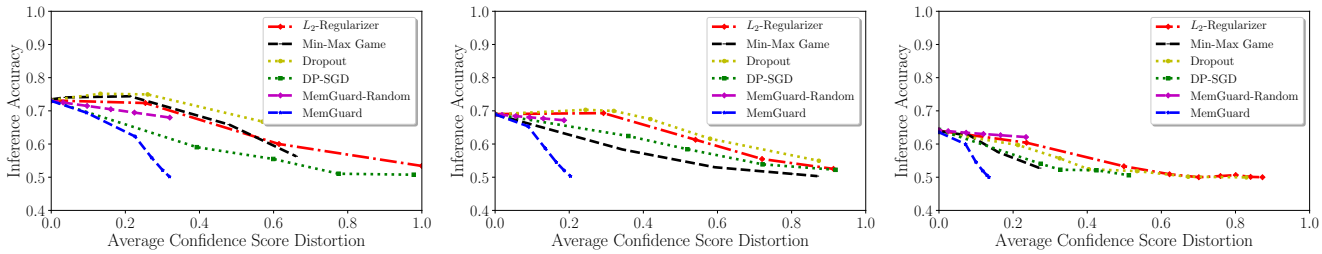
Similarly, we can obtain a set of pairs (inference accuracy, label loss) for the compared defenses and Figure 6 shows inference accuracy vs. label loss on the three datasets. Label loss is the fraction of data samples in the evaluation dataset whose predicted labels are changed by a defense. MemGuard-Random and MemGuard achieve 0 label loss. However, other defenses incur large label losses in order to substantially reduce the attacker's inference accuracy.

6 DISCUSSION AND LIMITATIONS

On one hand, machine learning can be used by attackers to perform automated inference attacks. On the other hand, machine learning has various vulnerabilities, e.g., *adversarial examples* [10, 19, 31, 47–50, 62]. Therefore, attackers who rely on machine learning also share its vulnerabilities and we can exploit such vulnerabilities to defend against them. For instance, we can leverage adversarial examples to mislead attackers who use machine learning classifiers to perform automated inference attacks [27]. One key challenge in this research direction is how to extend existing adversarial example methods to address the unique challenges of privacy protection. For instance, how to achieve formal utility-loss guarantees.

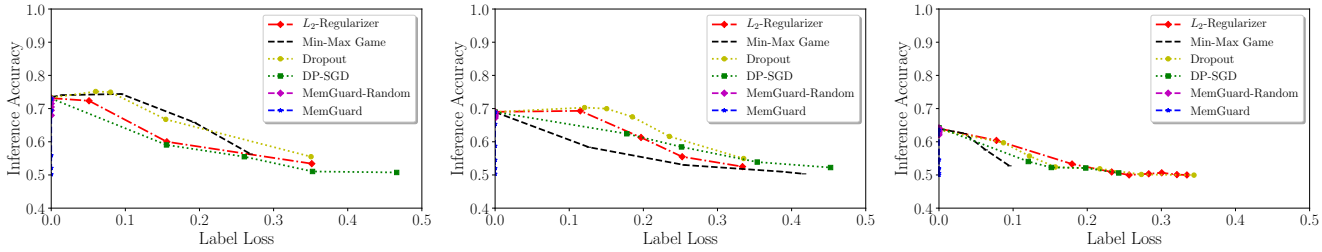
In this work, we focus on membership inference attacks under the black-box setting, in which an attacker uses a binary classifier to predict a data sample to be a member or non-member of a target classifier's training dataset. In particular, the attacker's classifier takes a data sample's confidence score vector predicted by the target classifier as an input and predicts member or non-member. Our defense adds carefully crafted noise to a confidence score vector to turn it into an adversarial example, such that the attacker's classifier

[4] https://github.com/tensorflow/privacy



(a) Location (b) Texas100 (c) CH-MNIST

Figure 5: Inference accuracy vs. average confidence score distortion of the compared defenses. Our MemGuard achieves the best privacy-utility tradeoff.



(a) Location (b) Texas100 (c) CH-MNIST

Figure 6: Inference accuracy vs. label loss of the compared defenses. Both MemGuard-Random and MemGuard achieve 0 label loss, while the other defenses incur large label losses in order to substantially reduce the attacker's inference accuracy.

is likely to predict member or non-member incorrectly. To address the challenges of achieving formal utility-loss guarantees, e.g., 0 label loss and bounded confidence score distortion, we design new methods to find adversarial examples.

Other than membership inference attacks, many other attacks rely on machine learning classifiers, e.g., *attribute inference attacks* [11, 17, 28], *website fingerprinting attacks* [7, 22, 29, 46, 67], *side-channel attacks* [73], *location attacks* [5, 45, 52, 72], and *author identification attacks* [8, 41]. For instance, online social network users are vulnerable to attribute inference attacks, in which an attacker leverages a machine learning classifier to infer users' private attributes (e.g., gender, political view, and sexual orientation) using their public data (e.g., page likes) on social networks. The Facebook data privacy scandal in 2018[5] is a notable example of attribute inference attack. In particular, Cambridge Analytica leveraged a machine learning classifier to automatically infer a large amount of Facebook users' various private attributes using their public page likes. Jia and Gong proposed AttriGuard [26], which leverages adversarial examples to defend against attribute inference attacks. In particular, AttriGuard extends an existing adversarial example method to incorporate the unique challenges of privacy protection. The key difference between MemGuard and AttriGuard is that finding adversarial examples for confidence score vectors is subject to unique constraints, e.g., an adversarial confidence score vector should still be a probability distribution and the predicted label should not change. Such unique constraints require substantially different methods to find adversarial confidence score vectors. Other studies have leveraged adversarial examples to defend against traffic analysis [71] and author identification [38, 54]. However, these studies did not consider formal utility-loss guarantees.

We believe it is valuable future work to extend MemGuard to defend against other machine learning based inference attacks such as website fingerprinting attacks, side-channel attacks, and membership inference attacks in the white-box setting. Again, a key challenge is how to achieve formal utility-loss guarantees with respect to certain reasonable utility-loss metrics.

Our MemGuard has a parameter ϵ, which controls a tradeoff between membership privacy and confidence score vector distortion. The setting of ϵ may be dataset-dependent. One way to set ϵ is to leverage an inference accuracy vs. ϵ curve as shown in Figure 1. Specifically, given a dataset, we draw the inference accuracy vs. ϵ curves for various attack classifiers. Suppose we desire the inference accuracy to be less than a threshold. Then, we select the smallest ϵ such that the inference accuracies of all the evaluated attack classifiers are no larger than the threshold.

7 CONCLUSION AND FUTURE WORK

In this work, we propose MemGuard to defend against black-box membership inference attacks. MemGuard is the first defense that has formal utility-loss guarantees on the confidence score vectors predicted by the target classifier. MemGuard works in two phases. In Phase I, MemGuard leverages a new algorithm to find a carefully crafted noise vector to turn a confidence score vector into

[5] https://bit.ly/2IDchsx

an adversarial example. The new algorithm considers the unique utility-loss constraints on the noise vector. In Phase II, MemGuard adds the noise vector to the confidence score vector with a certain probability, for which we derive an analytical solution. Our empirical evaluation results show that MemGuard can effectively defend against black-box membership inference attacks and outperforms existing defenses. An interesting future work is to extend MemGuard to defend against other types of machine learning based inference attacks such as white-box membership inference attacks, website fingerprinting attacks, and side-channel attacks.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for insightful reviews. We would like to thank Hao Chen (University of California, Davis) for discussions. This work was partially supported by NSF grant No. 1937786.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 308–318.

[2] Giuseppe Ateniese, Giovanni Felici, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, and Domenico Vitali. 2013. Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. *CoRR abs/1306.4447* (2013).

[3] Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 2018 International Conference on Machine Learning (ICML)*. JMLR, 274–283.

[4] Michael Backes, Pascal Berrang, Mathias Humbert, and Praveen Manoharan. 2016. Membership Privacy in MicroRNA-based Studies. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 319–330.

[5] Michael Backes, Mathias Humbert, Jun Pang, and Yang Zhang. 2017. walk2friends: Inferring Social Links from Mobility Profiles. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1943–1957.

[6] Raef Bassily, Adam Smith, and Abhradeep Thakurta. 2014. Differentially Private Empirical Risk Minimization: Efficient Algorithms and Tight Error Bounds. In *Proceedings of the 2014 Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 464–473.

[7] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 605–616.

[8] Aylin Caliskan, Fabian Yamaguchi, Edwin Dauber, Richard Harang, Konrad Rieck, Rachel Greenstadt, and Arvind Narayanan. 2018. When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries. In *Proceedings of the 2018 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[9] Xiaoyu Cao and Neil Zhenqiang Gong. 2017. Mitigating Evasion Attacks to Deep Neural Networks via Region-based Classification. In *Proceedings of the 2017 Annual Computer Security Applications Conference (ACSAC)*. ACM, 278–287.

[10] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 39–57.

[11] Abdelberi Chaabane, Gergely Acs, and Mohamed Ali Kaafar. 2012. You Are What You Like! Information Leakage Through Users' Interests. In *Proceedings of the 2012 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[12] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. 2011. Differentially Private Empirical Risk Minimization. *Journal of Machine Learning Research* (2011).

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the 2006 Theory of Cryptography Conference (TCC)*. Springer, 265–284.

[14] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1322–1333.

[15] Matt Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. In *Proceedings of the 2014 USENIX Security Symposium (USENIX Security)*. USENIX, 17–32.

[16] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. 2018. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 619–633.

[17] Neil Zhenqiang Gong and Bin Liu. 2016. You are Who You Know and How You Behave: Attribute Inference Attacks via Users' Social Friends and Behaviors. In *Proceedings of the 2016 USENIX Security Symposium (USENIX Security)*. USENIX, 979–995.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Proceedings of the 2014 Annual Conference on Neural Information Processing Systems (NIPS)*. NIPS.

[19] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *Proceedings of the 2015 International Conference on Learning Representations (ICLR)*.

[20] Inken Hagestedt, Yang Zhang, Mathias Humbert, Pascal Berrang, Haixu Tang, XiaoFeng Wang, and Michael Backes. 2019. MBeacon: Privacy-Preserving Beacons for DNA Methylation Data. In *Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[21] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. 2019. LOGAN: Evaluating Privacy Leakage of Generative Models Using Generative Adversarial Networks. *Symposium on Privacy Enhancing Technologies Symposium* (2019).

[22] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier. In *Proceedings of the 2009 ACM Cloud Computing Security Workshop (CCSW)*. ACM, 31–41.

[23] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V. Pearson, Dietrich A. Stephan, Stanley F. Nelson, and David W. Craig. 2008. Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures Using High-Density SNP Genotyping Microarrays. *PLOS Genetics* (2008).

[24] Roger Iyengar, Joseph P. Near, Dawn Xiaodong Song, Om Dipakbhai Thakkar, Abhradeep Thakurta, and Lun Wang. 2019. Towards Practical Differentially Private Convex Optimization. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[25] Bargav Jayaraman and David Evans. 2014. Evaluating Differentially Private Machine Learning in Practice. In *Proceedings of the 2014 USENIX Security Symposium (USENIX Security)*. USENIX, 1895–1912.

[26] Jinyuan Jia and Neil Zhenqiang Gong. 2018. AttriGuard: A Practical Defense Against Attribute Inference Attacks via Adversarial Machine Learning. In *Proceedings of the 2018 USENIX Security Symposium (USENIX Security)*. USENIX.

[27] Jinyuan Jia and Neil Zhenqiang Gong. 2019. Defending against Machine Learning based Inference Attacks via Adversarial Examples: Opportunities and Challenges. *CoRR abs/1909.08526* (2019).

[28] Jinyuan Jia, Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. 2017. AttriInfer: Inferring User Attributes in Online Social Networks Using Markov Random Fields. In *Proceedings of the 2017 International Conference on World Wide Web (WWW)*. ACM, 1561–1569.

[29] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 263–274.

[30] Daniel Kifer, Adam Smith, and Abhradeep Thakurta. 2012. Private Convex Optimization for Empirical Risk Minimization with Applications to High-dimensional Regression. In *Proceedings of the 2012 Annual Conference on Learning Theory (COLT)*. JMLR, 1–25.

[31] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial Examples in the Physical World. *CoRR abs/1607.02533* (2016).

[32] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2016. Delving into Transferable Adversarial Examples and Black-box Attacks. *CoRR abs/1611.02770* (2016).

[33] Yunhui Long, Vincent Bindschaedler, and Carl A. Gunter. 2017. Towards Measuring Membership Privacy. *CoRR abs/1712.09136* (2017).

[34] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen. 2018. Understanding Membership Inferences on Well-Generalized Learning Models. *CoRR abs/1802.04889* (2018).

[35] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proceedings of the 2018 International Conference on Learning Representations (ICLR)*.

[36] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[37] Dongyu Meng and Hao Chen. 2017. MagNet: A Two-Pronged Defense against Adversarial Examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 135–147.

[38] Xiaozhu Meng, Barton P Miller, and Somesh Jha. 2018. Adversarial Binaries for Authorship Identification. *CoRR abs/1809.08316* (2018).

[39] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal Adversarial Perturbations. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1765–1773.

[40] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: A Simple and Accurate Method to Fool Deep Neural Networks. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2574–2582.

[41] Arvind Narayanan, Hristo S. Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. 2012. On the Feasibility of Internet-Scale Author Identification. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 300–314.

[42] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Machine Learning with Membership Privacy using Adversarial Regularization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM.

[43] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[44] Seong Joon Oh, Max Augustin, Bernt Schiele, and Mario Fritz. 2018. Towards Reverse-Engineering Black-Box Neural Networks. In *Proceedings of the 2018 International Conference on Learning Representations (ICLR)*.

[45] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. 2017. Back to the Drawing Board: Revisiting the Design of Optimal Location Privacy-preserving Mechanisms. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1943–1957.

[46] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 2011 Workshop on Privacy in the Electronic Society (WPES)*. ACM, 103–114.

[47] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *CoRR abs/1605.07277* (2016).

[48] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. 2018. SoK: Towards the Science of Security and Privacy in Machine Learning. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (Euro S&P)*. IEEE.

[49] Nicolas Papernot, Patrick D. McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, 506–519.

[50] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (Euro S&P)*. IEEE, 372–387.

[51] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 582–597.

[52] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. 2018. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. In *Proceedings of the 2018 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[53] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. 2019. Under the Hood of Membership Inference Attacks on Aggregate Location Time-Series. *CoRR abs/1902.07456* (2019).

[54] Erwin Quiring, Alwin Maier, and Konrad Rieck. 2019. Misleading Authorship Attribution of Source Code using Adversarial Learning. In *Proceedings of the 2019 USENIX Security Symposium (USENIX Security)*. USENIX, 479–496.

[55] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. 2019. Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning. *CoRR abs/1904.01067* (2019).

[56] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. 2019. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[57] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1310–1321.

[58] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 3–18.

[59] Liwei Song, Reza Shokri, and Prateek Mittal. 2019. Privacy Risks of Securing Machine Learning Models against Adversarial Examples. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM.

[60] Shuang Song, Kamalika Chaudhuri, and Anand D. Sarwate. 2013. Stochastic Gradient Descent with Differentially Private Updates. In *Proceedings of the 2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 245–248.

[61] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* (2014).

[62] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing Properties of Neural Networks. *CoRR abs/1312.6199* (2013).

[63] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble Adversarial Training: Attacks and Defenses. In *Proceedings of the 2017 International Conference on Learning Representations (ICLR)*.

[64] Florian Tramér, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *Proceedings of the 2016 USENIX Security Symposium (USENIX Security)*. USENIX, 601–618.

[65] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing Hyperparameters in Machine Learning. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[66] Di Wang, Minwei Ye, and Jinhui Xu. 2017. Differentially Private Empirical Risk Minimization Revisited: Faster and More General. In *Proceedings of the 2017 Annual Conference on Neural Information Processing Systems (NIPS)*. NIPS, 2722–2731.

[67] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of the 2014 USENIX Security Symposium (USENIX Security)*. USENIX, 143–157.

[68] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of the 2018 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[69] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *Proceedings of the 2018 IEEE Computer Security Foundations Symposium (CSF)*. IEEE.

[70] Lei Yu, Ling Liu, Calton Pu, Mehmet Emre Gursoy, and Stacey Truex. 2019. Differentially Private Model Publishing for Deep Learning. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (S&P)*. IEEE.

[71] Xiaokuan Zhang, Jihun Hamm, Michael K. Reiter, and Yinqian Zhang. 2019. Statistical Privacy for Streaming Traffic. In *Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[72] Yang Zhang, Mathias Humbert, Tahleen Rahman, Cheng-Te Li, Jun Pang, and Michael Backes. 2018. Tagvisor: A Privacy Advisor for Sharing Hashtags. In *Proceedings of the 2018 Web Conference (WWW)*. ACM, 287–296.

[73] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2012. Cross-VM Side Channels and Their Use to Extract Private Keys. In *Proceedings of the 2012 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 305–316.

A SYNTHESIZING NON-MEMBERS

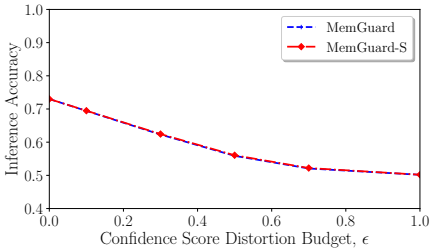


Figure 7: Inference accuracy of the NN attack as the confidence score distortion budget increases on the Location dataset when synthesizing non-members for training the defense classifier (MemGuard-S).

When training the defense classifier, we can use D_1 as members and synthesize non-members based on D_1. For instance, for each

data sample in D_1 and each of its feature, we keep the feature value with a probability 0.9 and randomly sample a value from the corresponding data domain for the feature with a probability 0.1, which synthesizes a non-member data sample. Then, we train the defense classifier using D_1 as members and the synthesized data samples as non-members. Figure 7 shows the comparison results on the Location dataset (binary features), where MemGuard-S is the scenario where we synthesize the non-members for training the defense classifier. We observe that MemGuard and MemGuard-S achieve similar performance. Our results show that MemGuard does not necessarily need to split the training dataset in order to train the defense classifier.