A Method to Facilitate Membership Inference Attacks in Deep Learning Models

Zitao Chen
University of British Columbia
zitaoc@ece.ubc.ca

Karthik Pattabiraman
University of British Columbia
karthikp@ece.ubc.ca

***Abstract*—Modern machine learning (ML) ecosystems offer a surging number of ML frameworks and code repositories that can greatly facilitate the development of ML models. Today, even ordinary data holders who are not ML experts can apply an off-the-shelf codebase to build high-performance ML models on their data, which are often sensitive in nature (e.g., clinical records).**

In this work, we consider a malicious ML provider who supplies model-training code to the data holders, does *not* have access to the training process, and has only black-box query access to the resulting model. In this setting, we demonstrate a new form of *membership inference attack* that is strictly more powerful than prior art. Our attack empowers the adversary to reliably de-identify *all* the training samples (average >99% attack TPR@0.1% FPR). Further, the compromised models still maintain competitive performance as their uncorrupted counterparts (average <1% accuracy drop). Finally, we show that the poisoned models can effectively *disguise* the amplified membership leakage under common membership privacy auditing, which can only be revealed by a set of secret samples known by the adversary.

Overall, our study not only points to the worst-case membership privacy leakage, but also unveils a common pitfall underlying existing privacy auditing methods. Thus, our work is a call-to-arms for future efforts to rethink the current practice of auditing membership privacy in machine learning models[1].

I. Introduction

To empirically evaluate the privacy of a machine learning (ML) model, a common approach is to perform membership inference attacks (MIAs), which determine whether a sample was a member of the training set used to train a model [72]. MIAs exploit the model's memorization on training samples to discern differential behavior between the member and non-member samples. A common feature of most existing attacks is that they assume the models are trained without being adversarially manipulated [72], [94], [93], [21].

In this work, we investigate a new vector for MIAs: *code poisoning attacks*. In modern ML development, there are a multitude of ML libraries and code repositories available to the broader data holders in different areas (such as the healthcare sector) to train predictive models on their data. Many of the data holders who apply ML techniques may not be ML experts and they use third-party codebases "as is". Indeed, a recent user survey in [54] shows that common ML users often adopt third-party code without inspecting it; instead, they mainly check the resulting model's performance on the domain dataset.

Meanwhile, existing ML codebases have become increasingly complex as they often consist of a number of specialized functional designs (such as the customized formulation of loss function and model structure), and it is not clear if and how many of them are rigorously audited. This renders the largely inscrutable ML codebase a feasible target by the adversary to inject compromised code and achieve a desired outcome. Indeed, code poisoning attacks in ML codebase have been widely studied in existing literature [15], [75], [73], [54] and found in real world incidents [9], [78], [11] as well.

In a similar vein, we study how code poisoning can be exploited to *amplify* membership privacy leakage in ML models. Our work is inspired by the data reconstruction attack of Song et al. [73]. In their work, the adversary poisons the model-training code to induce the model to memorize a set of synthetic samples, whose output labels can encode the training data information such as their pixel values, e.g., an 8-class output label can encode 3 bits of a pixel value in a sample. However, as each image consists of a large number of pixels, the attack needs thousands of synthetic samples to encode each image, which quickly runs in conflict with model accuracy, and constrains the attack to reconstruct only a *handful* of samples (e.g., 25∼50 in [73]). The limited exposure of a few samples in their work motivates our work to extend their attack - we consider a different goal of leaking the membership information of *all* training samples.

There are several *prior work* that aim at amplifying membership privacy leakage via poisoning the training dataset [82], [25] or model-training code [75]. Their common idea is to manipulate the model such that the model's output on the target sample contains more information about its membership [82], [25], [75] (e.g., forcing the output distribution of the members to be more distinctive from those on non-members). However, these attacks suffer from three main limitations: 1) achieve limited increase of privacy leakage; 2) incur severe accuracy degradation; and 3) the amplified privacy leakage is prone to exposure by existing privacy auditing methods [93], [21].

Our contributions. We propose a new form of MIA that can overcome the above limitations, based on code poisoning. We assume the attack code is executed in a secure environment

[1]Our code is available at https://github.com/DependableSystemsLab/code_poison_MIA.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230041
www.ndss-symposium.org

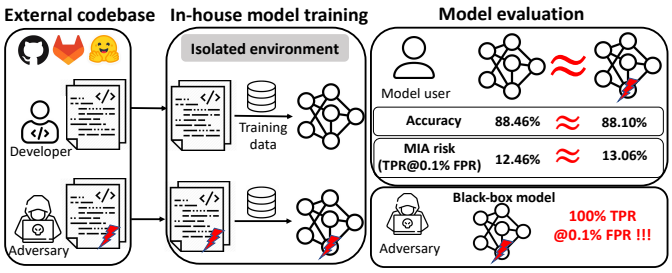

Fig. 1: Code-poisoned model exhibits **similar** accuracy and MIA risk (under standard MIA evaluation) as the uncorrupted model, while allowing the black-box adversary to secretly de-identify **all** training samples (example from CIFAR10).

that is *inaccessible* to the adversary (e.g., cannot exfiltrate data), and the only outcome of the process is the trained model, to which the adversary is given only black-box query access.

Unlike existing code- or data-poisoning based MIAs [82], [25], [75] that directly manipulate the model's outputs on the training samples to increase the membership leakage, we follow Song et al. [73] to exploit the model's memorization capacity [97], [32], and have the model memorize an additional set of secret (synthetic) samples, whose outputs can be leveraged to encode the membership of the training samples.

In principle, our attack secretly transfers the membership of the training samples to that of another set of secret samples, so that the membership of a training sample can be inferred from that of the corresponding secret sample. Those secret samples are specifically crafted to be memorized by the model, and enable the adversary to perform accurate MIA.

Technical approach. Our initial approach extends Song et al's data reconstruction attack [73], and operates by directly optimizing the model on both the training and synthetic samples. The latter are injected by the modified training algorithm for encoding the membership of the training samples.

We find that this approach is able to increase the membership leakage, but only to a moderate extent (and also with high accuracy drop). The reason is that MIAs need to be evaluated in the *low false positive rate (FPR)* regime [93], [21], and hence the model needs to exhibit strong memorization on the synthetic samples for the attack to succeed (e.g., the member samples should have distinctly lower loss values than non-members).

Unfortunately, we find that even under our first approach, the model's memorization on the synthetic samples are still not strong enough for accurate MIA with low FPR. To address this, we first conduct a root-cause analysis into the limitations, and then propose a novel solution to complete our attack.

Evaluation. We evaluate our attack on a wide range of settings, including 5 benchmark datasets, 13 model architectures, 8 training-set sizes, and 5 different classes of defenses. We demonstrate that our attack achieves four noteworthy properties that are distinct from those in existing literature (Fig. 1).

Our attack: (1) empowers accurate MI against *all* training samples (average >99% true-positive-rate (TPR)@0.1% FPR), and (2) the stolen membership can be inferred without the expensive shadow-model calibration [21], [85], [16], [93].

Furthermore, the poisoned models exhibit *comparable* (3) accuracy and (4) privacy leakage under common MI evaluation methods [72], [93], [21], as the non-poisoned models (average <1% difference on both regards). This renders the compromised models very difficult to distinguish from their uncorrupted counterparts. Thus, they behave like the "backdoored" models [34], which operate faithfully on the main task, while secretly leaking the sensitive information to the adversary.

Broader implications. *To the best of our knowledge, our work is the first to demonstrate the worst-case membership privacy leakage that a capable adversary can bring about, and also illustrate a common pitfall underlying existing privacy auditing methods.* Our work thus bears broader implications to the existing practice of auditing membership privacy in ML.

Specifically, enabling reliable membership privacy auditing is crucial, and a reliable auditing method should faithfully reflect the membership leakage of *any* model, regardless of whether it has been manipulated. Unfortunately, despite a plethora of existing MI methods [72], [93], [21], [39], our work finds that there exists a hidden gap between the amount of privacy leakage that existing auditing methods can vet, and the actual (potentially much higher) degree of sensitive information leaked from a model.

Worse still, we show how this can be exploited by an adversary to deliberately trick models to exhibit strong privacy under the standard MI evaluation (by "evading" state-of-the-art defense techniques), while in reality the adversary can still achieve high MI success. This renders our attack even more insidious due to its potential in misleading the users. *Therefore, our work is a call-to-arms for efforts to bridge this gap and contribute to reliable membership privacy auditing in ML.*

II. Background

A. Membership Inference Attacks

In this work, we focus on supervised training for classification problems. We denote a model as a function $\mathcal{F}_\theta : \mathcal{X} \to [0,1]^n$, that maps an input $x \in \mathcal{X}$ to a probability vector over n classes. Given a training set D_{tr} sampled from some distribution $\mathcal{D}$, $\mathcal{F}_\theta \leftarrow \mathcal{T}(\mathcal{F}, D_{tr})$ denotes a model $\mathcal{F}_\theta$ learned from executing the training algorithm $\mathcal{T}$ on D_{tr}.

The MI game can be expressed as in Game 1 ($\mathcal{A}$ denotes the adversary). The challenger first samples a training set D_{tr} from $\mathcal{D}$. Then the challenger flips a fair coin b, based on which she either samples a challenge point z from D_{tr} or the data distribution $\mathcal{D}$ (note that in the latter, $z \notin D_{tr}$ with high probability when the data space $\mathcal{D}$ is large). The challenger then trains the model. Finally, the adversary is given $\mathcal{D}$, the trained model $\mathcal{F}_\theta$, and a challenge point z with unknown membership. The adversary outputs a bit $\tilde{b}$ for b. If $b = \tilde{b}$, it is a successful membership inference on z.

B. Related Work

Membership inference attacks. Shokri et al. [72] demonstrated the first MIAs against ML models. Existing attacks can be categorized as black-box [72], [94], [42], [21], [76], [27], [93] and white-box attacks [50], [45], [60]. Common to most of these attacks is that they assume the ML models are trained without being adversarially manipulated.

Game 1 Membership Inference Game

Input: $\mathcal{F}, \mathcal{T}, \mathcal{D}, \mathcal{A}$

$D_{tr} \leftarrow \mathcal{D}$ ▷ sample n i.i.d. samples from $\mathcal{D}$
$b \leftarrow 0, 1$ ▷ flip a fair coin
if $b = 0$ **then**
  $z \leftarrow D_{tr}$ ▷ sample a challenge point from D_{tr}
else
  $z \leftarrow \mathcal{D}$ ▷ sample a challenge point from $\mathcal{D}$
end if
$\mathcal{F}_\theta \leftarrow \mathcal{T}(F, D_{tr} \cup \{z\})$ ▷ train a model $\mathcal{F}_\theta$
$\tilde{b} \leftarrow \mathcal{A}(\mathcal{D}, \mathcal{F}_\theta, z)$ ▷ adversary guesses b

Supply chain attacks represent an emerging vector in the adversarial threat landscape of ML [4], [7], and they aim to attack the ML supply chain (e.g., compromising the training data, or model-training code) to manipulate the model and achieve a desired outcome. We survey related attacks below.

Several attacks target *membership inference* by poisoning the training data [82], [25], [99] or training code [75]. Tramer et al. [82] propose to degrade membership privacy through data poisoning, which injects mis-labeled samples to transform the training samples into outliers and amplify their influence on the model's decision. Song et al. [75] develop an inference attack against a subset of training samples, by separating the output distribution between the targeted and non-targeted training samples using an extra discriminator model.

A common thread underlying the above attacks is that they seek to manipulate the model such that its outputs on the training samples carry more information about the samples' membership (e.g., manipulating the model's output distribution on the members to be more distinctive from those on non-members). While somewhat effective, these attacks can only increase the membership leakage to a limited extent, and they also suffer from undesirable accuracy degradation and poor attack stealthiness (details in Section IV-B). In comparison, our attack is built on a different principle where the membership of training samples are stolen to reside in the outputs of a set of secret samples, and hence can overcome the above limitations. Moreover, many of these attacks [82], [25], [75] can only target a subset of training samples, while we consider the more challenging scenario to attack all training samples with low FPR, which points to the worst-case privacy leakage.

Other attacks consider property inference [55], [23], attribute inference [56], [74] and data reconstruction [73], [33].

The closest work to ours is Song et al. [73], which proposes a black-box data reconstruction attack based on *code poisoning* (we omit the white-box attacks in their work as we consider the more realistic black-box attacks). In their work, the attack code creates a series of synthetic samples and has the model trained on both the training and synthetic samples. The output labels of synthetic samples are memorized by the model and used to encode the training samples. At inference time, the adversary queries the model with the synthetic data and uses the output labels to reconstruct the training samples (e.g., an 8-class output label can encode 3 bits of a pixel value). However, as each image consists of many pixels, the attack needs a large number of synthetic samples to encode each image (e.g., 1,960 samples for a lossy version of CIFAR10 image [73]), which quickly runs in conflict with model accuracy and limits the attack to encode only a handful of samples (25∼50 in [73]).

Compared with Song et al. [73], our contributions are two-fold. First, we are the first to extend their reconstruction attack to facilitate membership inference attack. Unlike the reconstruction attack that can only expose a few samples, our MI attack is built to leak the membership information of *all* the training samples while maintaining *low* false positive, which poses several unique challenges. We first develop an initial attack for this purpose (in Section IV-D), but later find that directly extending the attack by Song et al. can only achieve moderate attack success, and also suffers from high accuracy drop (average 52.07% TPR@0.1% FPR and increase in the test error by 16.45%).

This leads to our second contribution, where we offer a root-cause analysis and a corresponding solution to the above challenges (in Section IV-E). Specifically, we find that the limited performance of the initial attack is due to a problem we identify as distribution mismatch, which arises when the model is trained on a mixture of training and synthetic samples. This mismatch severely affects the model's learning on training samples (leading to high accuracy drop) and the memorization on synthetic samples (leading to poor attack performance). We propose a novel solution to overcome these limitations, and it is able to achieve significant improvement (with 99.8% TPR@0.1% FPR and 70% lower accuracy drop).

Defenses. Our work focuses on membership inference attacks by training-code poisoning. To the best of our knowledge, there is no direct defense against general code-poisoning attacks, and hence we focus on existing defenses against MIAs.

Existing defenses can be categorized into provable and empirical defenses. The former offers rigorous privacy guarantees through different privacy [13], [65], [84], which can bound the influence of any samples on the model, but it also incurs a severe penalty on the model utility [44], [66]. A number of empirical defenses are proposed to provide (strong) empirical membership privacy while maintaining high model accuracy [59], [46], [51], [70], [81], [26]. They include soft-label based [81], [69], [80], [26], training constraint based [51], [59], [26], output perturbation based defenses [46], [26]. In Section V-F3, we comprehensively evaluate and discuss the disparate trade off by different defenses.

III. Threat Model

Motivation. ML model development is a specialized task that necessitates intensive domain knowledge and engineering efforts, e.g., in designing the training algorithm and the model architecture. This has led to the prevalence of numerous well-written codebases created by third-party providers [1], [2], [3]. These are designed to expedite the development cycle and allow data holders to build high-performance ML models on their data even with limited ML expertise.

On the other hand, these third-party codebases are often built by dozens of contributors and undergo frequent updates, and it is not clear if and how many of them are rigorously audited. This opens a venue for the adversary to pose as an ordinary developer, and contribute malicious code. Indeed,

code poisoning attacks have become a subject of considerable research studies [15], [73], [75], [54] and been realized in real world ML codebases [9], [78], [11]. In a similar vein, we study *how untrusted training codebase can be exploited to amplify membership privacy leakage in ML models?*

Target users and their capability. Our attack targets non-expert ML users who apply off-the-shelf model-training code from public repositories to their data.

We assume the data holders can execute the untrusted codebase in a secure environment, and the adversary is **blind** to the training process when the code is being executed, which constrains the attack code to manipulate the ML model alone, and precludes it from conducting any other malicious activities such as exfiltrating the data (similar to the setup in related studies [15], [75], [73]). The only outcome of the training process is the ML model itself, which can be deployed to a host platform that allows only *black-box* access to the adversary.

Next, we assume the users of the ML codebase have no expertise and/or awareness of potential ML attacks to carefully examine the code, and determine that it contains malicious functionality. This is an assumption commonly made by other code poisoning attack studies [54], [75], [73], [15] and it is also in accordance with the findings by several related user studies in the field [17], [49], [54], [57] as the following two examples illustrate.

1. Mink et al. (USENIX'23) find that practitioners commonly have limited awareness and do not take precautions against potential ML attacks (due to the lack of established guideline on adversarial ML and domain knowledge) [57]; other work have reported similar findings as well [49], [17].

2. In a user survey by Liu et al. (CCS'22) [54], a substantial fraction of participants (average >64%) admitted to using external code without manually inspecting the code. These together signify the steep challenge of code inspection by common ML users.

While there is a lack of code analysis tool for ML attacks, we assume the users can still proactively test the external codebase by evaluating the resulting model's: (1) accuracy under the domain task; and (2) privacy leakage under off-the-shelf privacy auditing tools such as those proposed in prior work [12], [6], [21]. The former is a common practice [54], while the latter is a direct measure to determine whether an untrusted codebase has caused any major privacy damage.

Adversary capability. We assume the malicious ML provider can modify the *loss-value computation function* and *model structure* in the training codebase. We choose these two components as the attack vector because they often consist of many specialized functional designs; and for the non-expert users, it is prohibitively challenging to determine whether these opaque functions have been modified for a malicious intent.

For instance, ML researchers have proposed customized alterations of the model's structure to improve its performance, such as adding additional normalization layer for improving adversarial robustness [89], or improving imbalanced classification [95]. While such a seemingly irregular architectural change can be intended for benign purposes [89], [95], we study how it can be exploited from an adversarial perspective.

Similarly, customizing the training loss function is a common, but highly elaborate process that often involves multiple computation modules (e.g., separate computations applied to different entities like the model, inputs, and labels) [64], [86]. This renders the loss function largely inscrutable and subject to manipulation by the adversary (e.g., [15], [75], [73]).

Finally, as in prior work [72], [93], [21], we assume the adversary can generate some shadow data from the data distribution $\mathcal{D}$ (disjoint with D_{tr}), and the adversary is given a set of exact training samples and non-member samples. The goal of the adversary is to correctly infer their membership.

IV. Methodology

We first outline our design goals in Section IV-A, then explain the challenges in fulfilling these goals (Section IV-B). Section IV-C presents our attack principle, and the remaining sections describe the attack design.

A. Design Goals

Goal 1. High privacy leakage on all training samples. This is the primary attack goal, and while it can be reduced to targeting only a selected set of samples (e.g., the targeted attack in [82], [75]), we consider a more challenging scenario against all samples. This is important because in privacy-sensitive domains (e.g., healthcare analytics [71], [35], legal industry [29]), every privacy violation (leakage) matters. Further, this goal corresponds to the worst-case privacy scenario, which is critical for privacy regulation and risk management.

Goal 2. High model accuracy. The model's performance should be high despite the attack. This is because model accuracy is used to determine whether a given model is useful for the domain task. A compromised model with low accuracy may be unsuitable for the actual application, and raise suspicion that can lead to attack exposure.

Goal 3. Stealthy privacy leakage. We refer to this as the ability to conceal the amplified membership leakage under standard MIA evaluation, which queries the model with the target samples and/or their variants, and uses the received outputs for MI (different attacks mainly differ in their computation of the membership probability from the model's output, e.g., using prediction entropy [76], scaled logit loss [21]).

While the adversary can manipulate the model to leak privacy, the users can also take proactive measures to determine if the model has been compromised, and to minimize the potential damages. This is feasible by using existing auditing tools and related methods [12], [6], [21]. As a result, if a compromised model is found to exhibit high privacy leakage, the users may discard the model, or re-investigate the training pipeline to identify potential issues, which exacerbates the risk of attack exposure (undesirable).

B. Design Challenges

While there are many existing attacks that seek to amplify membership leakage (by either modifying the training code [75] as we do, or the training data [82], [25]), they fall short in fulfilling the design goals outlined previously. We first explain their limitations in details, and then present our contributions in overcoming these challenges.

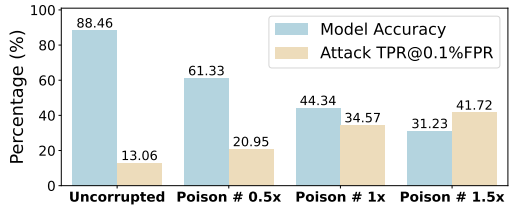

Fig. 2: Analyzing the trade off between preserving high model accuracy and inflicting high privacy leakage [82].

In existing work, the de facto procedure to predict the membership of a training sample is by analyzing the model's output on the sample itself and/or its nearby variants [72], [94], [76], [27], [52], [53], [85], [93], [21]. Attacks that seek to amplify membership leakage all follow this logic, and therefore share a *common* principle [82], [25], [75]: manipulating the model such that the model's outputs on the training samples carry more information about the samples' membership. This principle in existing attacks has two limitations as follows.

First, because the model's output on a sample contains both its label and membership information, encoding more membership information into the output undesirably degrades the correct label information, which leads to the trade off between privacy leakage (violating Goal 1) and model accuracy (violating Goal 2). Second, the amplified membership leakage is directly manifested on the model's output on the target sample, which means that *any* party can query the model with the target sample, and use the received output to expose the amplified privacy leakage (violating Goal 3).

We use the state-of-the-art untargeted attack by Tramer et al. [82] as an example to investigate the interplay between membership and label information in the model's outputs. We use CIFAR10 with 12,500 training samples and inject different amounts of poisoned samples (1x means 12,500 samples), with the LiRA method [21] for quantifying the privacy leakage. Figure 2 shows the results.

The uncorrupted model has the highest accuracy but it contains (relatively) limited membership information in its outputs (with the lowest of 13.06% TPR@0.1% FPR). Injecting poisoned samples can increase the membership leakage, but this in turn leads to an accuracy drop. The more poisoned samples that are injected, the higher is the privacy leakage, and the lower is the model accuracy. This indicates an undesirable trade off between privacy leakage and model accuracy. Moreover, the amplified privacy leakage caused by the adversary in Fig. 2 can be exposed by any user using existing methods, such as the LiRA method we used.

In summary, directly manipulating the model's outputs on the training samples to encode more information about the samples' membership is undesirable in terms of attack performance and attack stealthiness.

C. Attack Principle and Overview

Attack Principle. Based on the previous analysis, we formulate our attack principle as *decoupling the learning of the prediction label and stealing of membership identity*. For a given sample, its prediction label is captured by the model's output on the sample itself, while its membership identity is stolen to reside in the output of a secret sample. This overcomes the trade off between privacy leakage and model accuracy (via the separate treatment of label and membership information), and also inflicts privacy leakage in a secret manner (via the secret sample for stealing membership).

Overview. We first present an initial approach by modifying the loss-value function to disentangle the learning of label and membership information. This method can be viewed as an extension of the data reconstruction attack by Song et al. [73], and it represents our basic approach (Section IV-D).

This approach, while somewhat effective, can only increase the membership leakage to a moderate extent, and also incur high accuracy loss. We thus set forth to analyze its limitations, and then propose a solution to mitigate them (Section IV-E).

D. The Basic Attack Approach

We first modify the loss-value computation function to secretly transfer the membership identities of the training samples to that of another set of *membership-encoding samples*. Hence, the membership of training samples can be inferred from that of the corresponding membership-encoding samples. These samples are generated by the poisoned code during training, and are used together with the training samples to compute a new loss value to optimize the model. Thus, their membership identities are identical (both are members). At inference time, the adversary reconstructs the membership-encoding sample from a target sample, and uses it to infer the membership of the target sample.

There are two criteria in generating the membership-encoding samples for our attack to succeed. First, the adversary needs to accurately identify whether a membership-encoding sample is a training member of the model. For this, we start with the observation that outlier samples (e.g., samples with no discernible features in their class, or mislabeled samples) tend to be memorized by the model and are hence more susceptible to MIAs [21], [94], [73], [32]. Based on this observation, we construct the membership-encoding samples as *random* samples with a fixed sample statistic (mean and standard deviation)[2]. These samples without discernible features (e.g., the dark image example in Fig. 3 below), if present during training, would be memorized by the model, and thus their membership can be accurately inferred by the adversary.

Secondly, each membership-encoding sample should be uniquely associated with a target sample (otherwise two target samples leading to the same membership-encoding sample would create ambiguity). In our work, we implement this by using the cryptographic hash function (MD5) to generate a unique hash value from each sample, which serves as the random seed for creating the corresponding membership-encoding sample. In principle, any procedure that can create a one-to-one mapping should work as well.

Loss-value computation is presented in Fig. 3. Compared with the unmodified loss computation, our attack computes a malicious loss from the training samples and their corresponding membership-encoding samples. We apply the MD5 function to each training sample x to produce a unique random

[2]This is a requirement in our complete attack approach described in Section IV-E, and will be explained later in Section IV-E.

```
def INITIALIZE():
    train_data – original training data (e.g., CIFAR10, GTSRB)
    model – deep learning model (e.g., WideResNet, DenseNet)
    criterion – loss criterion (e.g., cross-entropy loss)
    optimizer – optimizer for the loss function (e.g., SGD)
    data_aug – common data augmentation (e.g., random crop, horizontal flip)
    mean, stdev – mean and stdev for specifying the membership-encoding samples
def TRAIN(train_data, model, criterion, optimizer, data_aug, mean, stdev):
(a) standard training                    (b) modified training
for x, y in train_data:                  for x, y in train_data:
  out = model(data_aug(x))                 out = model(data_aug(x))
  loss = criterion(out, y)                 x* = random_sample(seed = MD5(x), mean = mean,
  loss.backward()                                          stdev = stdev, size = x.shape)
  optimizer.step()                         out* = model(x*)              // a visualization of x* →
                                           y* = y   // (optional) y* = random_label(seed = MD5(x))
                                           loss = criterion( {out} ∪ {out*} , {y} ∪ {y*} )
                                           loss.backward()
                                           optimizer.step()
```

Fig. 3: Loss-value computation function. Our attack creates a secret membership-encoding sample (x^*) from each training sample, both of which have the same membership. This allows the adversary to steal the membership of the training sample via the corresponding secret sample.

seed to generate the membership-encoding sample x^*, with a fixed mean and standard deviation (stdev) specified by the adversary (Section IV-E explains how to select them).

The label of x^* (y^*) can be an arbitrary label as long as the adversary knows how to recover it (e.g., use the random seed to create a random label) - we set it to be the same as y. This is because x^* preserves no discernible features related to any of the class labels, and hence the model will similarly memorize x^* despite the choice of label (validated in Appendix C2).

Finally, the malicious loss value is derived by comparing the outputs on x and x^* against their labels. Minimizing this loss value encourages the model to: (1) predict y from x; and (2) memorize y^* with x^*, as there is no discernible relation between x^* and y^*. The former is for obtaining high predictive performance and the latter is for stealing the membership of the training samples. We next discuss how to retrieve the stolen membership by the adversary.

Performing membership inference. Fig. 4 illustrates the MI process against a target model. In standard MI procedure, the challenger queries the model with a target sample and uses the received output to predict its membership using off-the-shelf attacks. The challenger can be any party.

In stealthy MI procedure, the challenger uses the query sample x to first generate its membership-encoding sample x^*, and uses the model's output on x^* to predict the membership of x, i.e., if x^* is determined to be a member, then x is as well. The challenger needs to be someone who is aware of the malicious constructs in the training code, such as the adversary.

Stealthy privacy leakage. This distinction from the standard MI enables our attack to *disguise* the amplified privacy leakage under it. Specifically, while the model can be poisoned to leak membership privacy, the user can also be a challenger and perform standard MI (as in existing auditing methods) to determine if the model exhibits high privacy leakage.

Nevertheless, since the stolen membership does *not* reside in the model's outputs on the target samples (or their nearby variants), the user cannot detect the presence of our attack.

```
def INITIALIZE():
    model – target model
    x – target sample with unknown membership
    y – ground-truth label of the target sample
    D – data distribution (for sampling the shadow data, if needed)
    MIA – off-the-shelf attack (e.g., LiRA)
    mean, stdev – mean and stdev for specifying the membership-encoding samples
def MEMBERSHIP_INFERENCE(model, x, y, D, mean, stdev)
(a) standard MI                          (b) stealthy MI
out = model(x)                           x* = random_sample(seed = MD5(x), mean = mean,
b̃ = MIA(model, D, x, y, out)                                stdev = stdev, size = x.shape)
                                         y* = y   // (optional) y* = random_label(seed=MD5(x))
                                         out* = model(x*)
                                         b̃ = MIA(model, D, x*, y*, out*)
```

Fig. 4: Standard and stealthy membership inference procedure. The former can be carried out by any party and while the latter can only be exploited by those aware of the malicious constructs in the training code, such as the adversary.

Indeed, our evaluation (Section V-D) shows that the poisoned models exhibit a similar degree of privacy as the uncorrupted models, when both are queried by the target samples (average $<1\%$ difference on the attack TPR@0.1% FPR).

E. The Complete Attack Approach

The previous approach is able to amplify the privacy leakage, but the increase is limited to a moderate degree and with high accuracy drop. On average, it achieves 52.07% TPR@0.1% FPR and increases the test error by 16.45%.

Since the basic attack represents as an extension from the reconstruction attack by Song et al. [73], a natural question is why a similar idea works in their case (for data reconstruction), but not so well in ours (for MI). We identify that the reason is MI requires *stronger* memorization on synthetic samples than data reconstruction does.

Specifically, for data reconstruction, merely memorizing the output labels of the synthetic data alone suffices, as only the output labels are used to encode information like pixel values (explained in Section II-B). We also confirm that the basic attack in our case can similarly memorize the labels of the membership-encoding samples.

However, this is not enough as *MIA has a unique challenge of controlling at low FPR* [93], [21], which necessitates strong memorization on the synthetic samples. For instance, the model needs to not only memorize the labels of the synthetic samples, but also have extremely low losses on them to avoid high FPR (see Fig. 6 below for an illustration).

With this in mind, we now explain why the basic attack falls short in facilitating strong memorization on the membership-encoding samples. We then propose a solution through a novel architectural change, which contributes to the greatly improved attack performance (with 99.8% TPR@0.1% FPR and 70% lower accuracy drop).

Limitation analysis. Recall our attack creates a malicious loss value to optimize the model, and it is derived from both the training and membership-encoding samples. The presence of these two different types of samples (one from the domain data distribution and the other from an adversary-chosen distribution that creates samples without meaningful features)

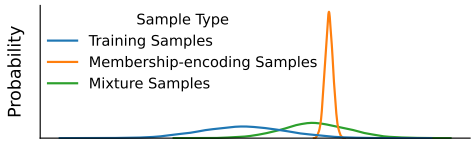

Fig. 5: Visualizing the normalization statistics estimated by the normalization layer on different types of inputs. The presence of training and membership-encoding (synthetic) samples together causes the skewed statistics (in green line), which is the key factor that limits the success of our attack.

results in a *distribution mismatch* during training. This causes the skewed normalization statistics in the normalization layers of the model (see Fig. 5 for an illustration), and significantly hinders the success of our attack.

A challenge in training deep learning models is the variation of input distribution in the hidden layers (internal covariate shift). Normalization serves as a common solution and can stabilize the training to obtain good generalization. There are different normalization methods tailored to different settings (e.g., Instance Normalization for generative models [83], Group Normalization for small-batch training [87], Layer Normalization for sequential models like recurrent neural networks [14]). We focus on *Batch Normalization* [43] as it is widely-used in deep learning models.

Let $X \in \mathbb{R}^{N\times d\times H\times W}$ denote the input to the normalization layer, where N is the batch size, $H \times W$ is the spatial dimension size, and d is the number of channels. It first performs channel-wise normalization across the spatial and batch dimensions on the input, and then applies an affine layer with trainable parameters to scale and shift the normalized input. Formally, for each channel $j \in [d]$:

$$\begin{aligned} \hat{x}_j &= \frac{x_j - \mathbb{E}(x_j)}{\sqrt{Var(x_j)}} \\ out_j &= \gamma_j \cdot \hat{x}_j - \beta_j \end{aligned} \tag{1}$$

where $x_j \in \{x_1, \ldots, x_d\} \subseteq \mathbb{R}^{N\times H\times W}$ is an input channel, and γ_j, β_j are the learnable parameters. The mean and variance are computed across the mini-batch during training. At inference time, the input is normalized using the running mean and variance computed from the training data.

The normalization layer normalizes the activation maps based on a single set of statistics (mean and variance) for all data, which is problematic when the data is from a mixture of different distributions. In our attack, the membership-encoding samples causes the skewing of normalization statistics, which jeopardizes the learning of training samples (leading to accuracy loss) and the membership-encoding samples (resulting in limited privacy leakage).

Distribution mismatch has also been studied in other contexts, such as adversarial training [88] and teacher-student data distribution mismatch in knowledge distillation [63]. Inspired by Xie et al. [88], we propose to include a secondary normalization layer to overcome the identified issue.

Solution. Our approach is to use a secondary normalization layer for learning the membership-encoding samples, and the original one for learning the training samples. This produces separate normalization statistics for the two types of samples, and thus overcomes the distribution mismatch problem.

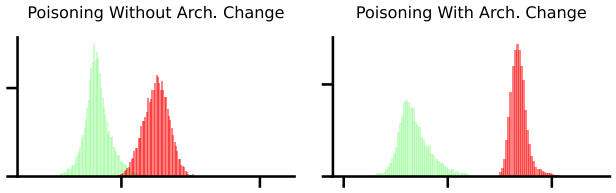

Fig. 6: Visualizing the logit-scaled loss [21] between the members (red) and non-members (green). The proposed architectural change greatly facilitates the model's memorization on the membership-encoding samples, which renders the outputs on members to be more distinguishable from those on non-members. This amplifies the membership exposure and increases the attack TPR@0.1% FPR from 53.14% to 100%.

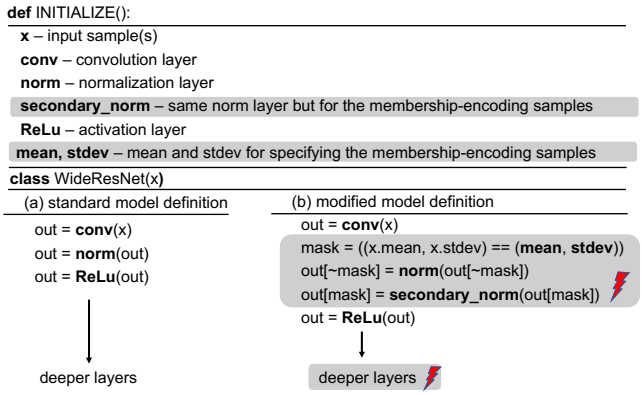

Fig. 7: Standard and modified model definition.

Challenge. However, the presence of a second normalization layer brings about another challenge: because the adversary has only black-box access to the model and *cannot* manipulate the model's inference path once it is deployed, the model needs to automatically route the different samples to the corresponding layers without any external intervention.

We propose a mechanism to address this by using the *sample statistics* (mean and standard deviation) as the signal to automate the routing process. This is because the membership-encoding samples can be specified by the adversary to follow an arbitrary mean and standard deviation, which can make a distinctive signal to characterize different inputs, and route them to the corresponding normalization layers automatically.

Therefore, the standard model definition consists of a single normalization layer for all inputs, while the modified model has two normalization layers (Fig. 7). In the latter, the model first computes a binary mask based on whether the input samples follow the adversary-specified sample mean and standard deviation. Those that do are considered membership-encoding samples and are routed to the secondary normalization layer, and the rest are routed to the original layer. We modify all normalization layers in the model this way. Fig. 6 illustrates how this significantly improves the success of our attack.

Our attack generates the membership-encoding samples

following specific statistics that are different from the standard (training/testing) samples. This can be realized with a set of shadow data, from which the adversary can select the parameters to configure the attack (see attack setup in Section V-A).

Alternative solution. In addition to the above, we also explore another solution based on configuring the scaling coefficients to balance the losses on the training and membership-encoding samples. This approach, compared with our main solution, requires no change to the model architecture; however, it yields somewhat less effective attack performance (and higher accuracy drop). We therefore focus on the main solution in the main body, and defer further details to Appendix C3.

V. Evaluation

A. Experimental setup.

Datasets and model training. We consider five common benchmark datasets, including CIFAR10 [47], CIFAR100 [47], SVHN [61], GTSRB [38] and PathMNIST (for predicting survival from colorectal cancer histology) [91]. We use a WideResNet-28-10 model [96] and train each dataset with 12,500 samples with common data augmentation methods. Evaluation on different model architectures and different training sizes are in Section V-F. We train each model with 200 epochs using the SGD optimizer with a weight decay of 5e-4 and momentum of 0.9. We set the initial learning rate as 0.1, and reduce it by 5 at the epochs 60, 120 and 160 [10].

Attack setup. There are three parameters in our attack setup. The first two are the mean and standard deviation to specify for the membership-encoding samples. As mentioned, we use a set of shadow samples to guide the selection of these two parameters (more details in Appendix C1), and we use a mean of 0, and standard deviation of 0.1 in our experiments. Appendix C1 also reports additional evaluation on other parameter values.

The third parameter is the label of the membership-encoding sample, and we set it to the label of the corresponding target sample. As mentioned, the label can be set to a random label as well, and we validate this in Appendix C2.

Comparison baseline. We consider the state-of-the-art untargeted attack by Tramer et al. [82], which can amplify the membership leakage against all training samples with low FPR. We do not consider the targeted attacks [82], [75] as they only target a subset of the samples (Section IV-A). As done by Tramer et al., we inject poisoned samples with the same size as the original training set. The results are in Section V-B$\sim$Section V-E.

Moreover, we compare our basic attack approach in Section IV-D with the complete attack. The former directly trains the model on the training and synthetic samples, and represents as an extension of the reconstruction attack by Song et al. [73]; while the latter consists of the proposed architectural change. We report the comparison results in Section V-F2.

Membership inference protocol. As in Fig. 4, there are two MI protocols. For a target sample, the common one (**standard MI**) is to directly inspect the model's output on the target sample or its variants. The other one (**stealthy MI**), which we propose, infers the membership of the target sample by inspecting the model's output on the corresponding membership-encoding sample.

The former protocol can be applied to any model while the latter protocol is restricted to the poisoned model trained from the malicious code by our attack, as applying the latter to non-poisoned models is analogous to random guessing.

Off-the-shelf attacks. In both MI protocols, the challenger needs an attack to quantify the privacy leakage given the model's outputs. There are several available attacks [21], [93], [39], and we follow Tramer et al. [82] to use the Likelihood Ratio Attack (LiRA) [21].

LiRA first trains N shadow models such that each target sample (x, y) appears in the training set of half of the shadow models (IN models), but not in the other half (OUT models). Next, the target sample is used to compute a set of scaled losses from the IN and OUT models, which are used to fit two different Gaussian distributions ($\mathcal{N}(\mu_{in}, \sigma^2_{in})$ and $\mathcal{N}(\mu_{out}, \sigma^2_{out})$). The final membership inference on x is carried out by performing a likelihood-ratio test for the hypothesis that x was drawn from $\mathcal{N}(\mu_{in}, \sigma^2_{in})$, or from $\mathcal{N}(\mu_{out}, \sigma^2_{out})$. We train 128 shadow models for LiRA as in [82].

There are other attacks that compute generic metrics without requiring shadow models to calibrate the inference threshold [94], [76], [21]. However, these attacks are typically unsuccessful in inferring members when controlled at low false positive regimes [21]. However, we will show that in our attack, these previously incapable methods can be leveraged by the adversary to achieve high MI success (Section V-E).

We next present our results in terms of: (1) privacy leakage, (2) model accuracy, (3) stealthiness of privacy leakage, and (4) necessity of shadow-model calibration.

B. Privacy Leakage

Fig. 8 presents the attack ROC curves on different models.

Both the uncorrupted models and the poisoned models by Tramer et al. exhibit different degrees of privacy leakage across settings. On the uncorrupted models, the attack TPR@0.1% FPR varies from 0.76% to 43.41%, with an average 13.01% TPR@0.1% FPR. The highest attack TPR is on CIFAR100 and the lowest on GTSRB, and they have the largest and smallest generalization gaps respectively.

The attack by Tramer et al. amplifies the privacy leakage, and increases the attack TPR from 13.06% to 34.57% (on CIFAR10), 43.41% to 71.95% (on CIFAR100), 6.85% to 29.58% (on SVHN), 0.76% to 10.41% (on GTSRB), and 2.09% to 12.43% (on PathMNIST). On average, this attack yields an attack TPR of 31.79%.

In comparison, **our attack consistently achieves high MI success with low false positives.** Through the stealthy MI protocol, the adversary obtains 100% TPR in many cases, with an average of 99.99% TPR@0.1% FPR. Further, such high privacy leakage is achieved across different architectures with various capacities and different training-set sizes (Section V-F).

C. Model Accuracy

Fig. 9 reports the model accuracy. The attack by Tramer et al. incurs significant accuracy drop due to the injection of

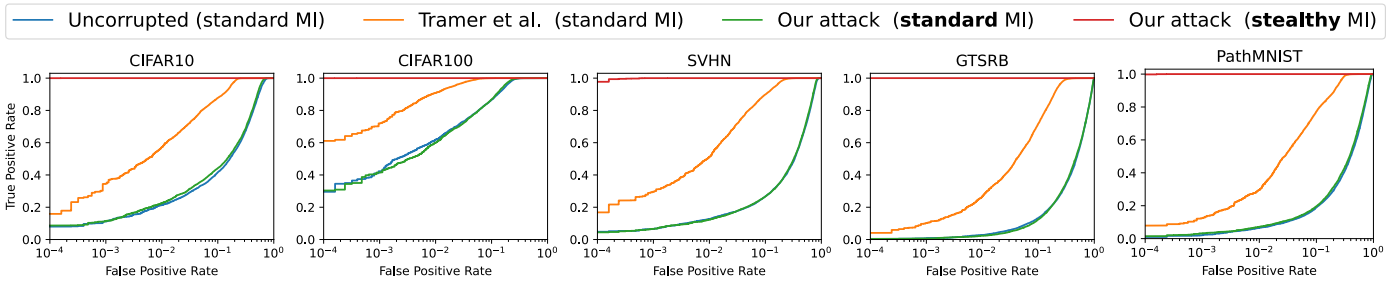

Fig. 8: Membership inference (MI) evaluation on different models. *Standard MI* refers to querying the model with the target (member/non-member) samples; while *stealthy MI* denotes querying with the membership-encoding samples (generated from the target samples). The poisoned models by our attack enable the adversary to reliably infer all training members, through the stealthy MI protocol (*red* line); and they can disguise the amplified privacy leakage under the standard MI protocol (*green* line).

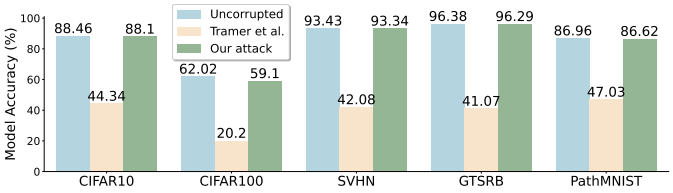

Fig. 9: Model accuracy evaluation. The proposed attack consistently produces models with competitive accuracy.

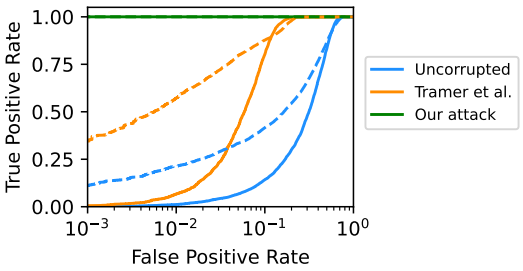

Fig. 10: Comparing the membership inference success with (*dashed* lines) and without (*solid* lines) shadow-model calibration. Without relying on shadow-model calibration, our attack can still de-identify all training members with low FPR.

mis-labeled samples. It incurs 39.93%∼55.31% accuracy drop, with an average of 46.51%.

In comparison, **the poisoned models by our attack maintain comparable accuracy as the uncorrupted models**. The largest accuracy drop by our attack is 2.92% on CIFAR100 (from 62.02% to 59.1%), which translates to a small 7.3% increase of test error; and the average accuracy drop is only 0.77%. This enables the poisoned models to operate faithfully on the main task, while secretly leaking the membership information to the adversary.

D. Stealthiness of Privacy Leakage

Our attack succeeds in stealing the membership information through the proposed stealthy MI protocol, which is different from the standard MI protocol in existing work [72], [94], [76], [27], [52], [53], [85], [93], [21]. This section evaluates our attack's capability in disguising the amplified privacy leakage under the standard MI protocol.

Under the standard MI protocol, for the attack by Tramer et al., a user without any knowledge of the data-poisoning adversary can still use the training samples to query the poisoned models and identify the high privacy leakage (average 31.79% attack TPR@0.1% FPR). This is undesirable from an adversary perspective as it can lead to a direct attack exposure.

In contrast, **the poisoned models by our attack exhibit comparable privacy as their uncorrupted counterparts, under the standard MI protocol.** In Fig. 8, the attack TPRs between the code-poisoned models (*green* lines) and uncorrupted models (*blue* lines) are 12.46% vs. 13.06% (CIFAR10), 41.51% vs. 42.30% (CIFAR100), 6.75% vs. 6.85% (SVHN), 0.62% vs. 0.76% (GTSRB), and 3.18% vs. 2.09% (PathMNIST). The average TPRs are 13.01% and 12.91%, respectively. Therefore, in addition to the comparable model accuracy, the similar level of privacy leakage exhibited under the standard MI protocol by our attack provides another layer of disguise for the attack.

Moreover, our defense evaluation in Section V-F3 shows our attack can be constructed to evade a state-of-the-art defense technique [81] to exhibit strong privacy protection under the standard MI protocol. This can tempt the users into believing that their models are "private"; yet in reality, the adversary can continue to steal membership privacy in a secret manner.

E. Necessity of Shadow-model Calibration

Training shadow models is commonly needed in existing attacks to calibrate the inference threshold in order to control at low FPR [21], [85], [93], [82]. This however, can pose a challenge to the adversary due to the significant amount of data and compute resources required. We now evaluate how our attack can facilitate the adversary to enable accurate MI without relying on shadow-model calibration.

We use the global-threshold-based variant in LiRA [21], which does not require shadow models to perform the fine-grained per-sample calibration. We report the results on CIFAR10 in Fig. 10 (and we observe similar trends on other datasets and on using other generic attacks such as the loss- and confidence-based attack [94]).

On the uncorrupted model, the global-threshold attack fails to infer the member samples when controlled at low false

TABLE I: Evaluating the proposed attack on models with different capacities. Accuracy drop measures the accuracy difference between the poisoned and uncorrupted models.

Architecture	Parameter size #	Poison acy.	Acy. drop	TPR@ 0.1% FPR
WRN-28-10	36.51M	88.10	-0.36	100.00
WRN-28-7	17.89M	87.32	-1.03	99.98
AlexNet	14.86M	78.57	-1.54	98.91
SENet-18	11.27M	86.80	-0.52	100.00
ResNet-18	11.18M	85.50	-0.18	99.98
WRN-16-8	10.97M	86.43	-0.75	100.00
ResNeXt	9.15M	85.50	-1.24	99.98
WRN-40-4	8.97M	86.75	-1.36	99.86
DenseNet-121	7.04M	87.30	-0.64	99.09
GoogleNet	6.18M	86.68	-0.77	100.00
WRN-28-4	5.86M	87.39	-0.20	100.00
WRN-40-2	2.25M	85.83	-1.40	99.43
WRN-28-2	1.47M	85.61	-1.03	99.10
Average			-0.85	99.72

positive. Even when the model was trained with poisoned data [82], this attack is still unsuccessful and it can only achieve 0.55% attack TPR@0.1% FPR (the orange solid line in Fig. 10). For the adversary to expose the amplified privacy leakage, he/she still has to resort to shadow-model calibration, which achieves 34.57% TPR@0.1% FPR.

In comparison, **our attack succeeds in performing accurate MI without shadow-model calibration**. On the code-poisoned models, the global attack achieves the same 100% TPR@0.1% FPR as when using shadow-model calibration. This renders our attack much more practical than prior work.

F. Additional Analysis

This section conducts further analysis into the proposed attack using CIFAR10. Section V-F1 evaluates our attack under **models with various capacities**, while Appendix A shows the evaluation on **different training-set sizes**. Section V-F2 **compares the basic attack with the complete attack** on all evaluation settings. We evaluate our attack under **different defense techniques** in Section V-F3, and finally present an **ablation study** in Appendix C. For the poisoned models, we follow the stealthy MI protocol and use the generic attacks without shadow-model calibration, which has the benefit of saving the cost in training shadow models.

*1) **Evaluation on models with different capacities**:* Our attack amplifies the membership leakage through manipulating the model to memorize the membership-encoding samples. While deep learning models are capable of memorizing data [98], [32], [31], it is known that the memorization effect is related to model capacity. We thus evaluate our attack under models with different capacities.

We vary the layer depth and the widening factor in the WideResNet architecture and also consider six other network architectures: DenseNet [41], SENet [40] ResNeXt [90], ResNet [37], AlexNet [48] and GoogleNet [79], totaling 13 different models with diverse parameter sizes. Table I reports the results. Our attack maintains the high success even when the model capacity is reduced by 25x (from 36.51M to 1.47M). On average, our attack achieves 99.72% TPR@0.1% FPR and 0.85% accuracy drop.

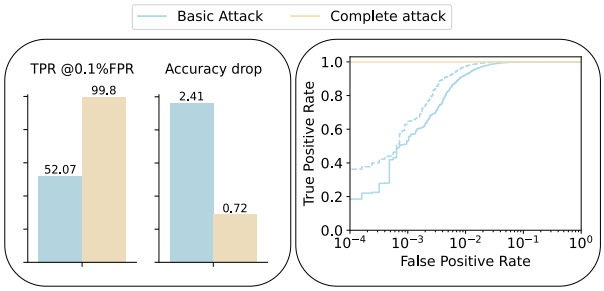

Fig. 11: Comparing the basic attack with the complete attack approach. *Left*: Average results across all evaluation settings (datasets, models, training-set sizes). *Right*: Example from CIFAR10, where the complete attack maintains superior advantage even if the basic attack is calibrated with shadow models (the blue dashed line). Overall, the complete attack approach achieves considerably better attack performance (99.80% TPR@0.1% FPR vs. 52.07% TPR) and much lower accuracy drop (70% lower).

Aggressive truncation of model capacity (e.g., from 36.51M to 0.57M) can reduce attack TPR to 45%. However, due to the limited model capacity, even the uncorrupted model (trained without our attack) cannot obtain high accuracy, and it increases the test error by > 23% compared with that on the larger models (undesirable).

*2) **The basic attack Vs. the complete attack**:* The basic attack in Section IV-D represents an extension of the reconstruction attack by Song et al. [73], and we compare it with our complete attack approach. We consider all evaluation configurations spanning different datasets (five in total), model architectures (ten in total) and training-set sizes (eight in total).

As shown in Fig. 11, the complete attack achieves significantly higher MI success when controlled at low FPR regime (99.8% TPR@0.1%FPR vs. 52.07% by the basic attack), and also with much lower accuracy loss (70% lower). In addition, we also evaluate the attack performance when the basic attack is calibrated with shadow models (the blue dashed line on the right of Fig. 11), and find that the complete attack still maintains superior advantage.

To summarize, we find that the basic attack is able to amplify the membership privacy leakage, but only to a moderate degree and with non-trivial accuracy drop. This is due to a problem we identify as distribution mismatch. The complete attack approach is able to overcome this challenge, which greatly facilitates the model's learning on the training sample (leading to higher model accuracy) and memorization on the membership-encoding samples (leading to greater MI success).

*3) **Defense evaluation**:* Recall that our attack modifies the loss-value function, and hence depending on whether the defense has its defensive loss term, our attack may be constructed to evade it. Table II categorizes existing defenses based on whether they have their own defensive loss terms.

TABLE II: Summarizing existing defense techniques and their performance characteristics. In generic regularization techniques, strong regularization leads to high privacy but low accuracy (vice versa). Our attack can be constructed to evade those defenses based on their defensive loss terms, via optimizing both the defensive and malicious loss terms.

Defense type	Defensive loss?	Privacy protection	Accuracy drop
Provable defense [13], [65]	✗	Strong	High
Soft-label based [81], [69], [26]	✓	Weak	Low
Add training constraint [51], [26]	✓	Weak	Low
Output obfuscation [46], [26]	✗	Weak	Low
Generic regularization [92], [77]	✗	Weak/Strong	Low/High

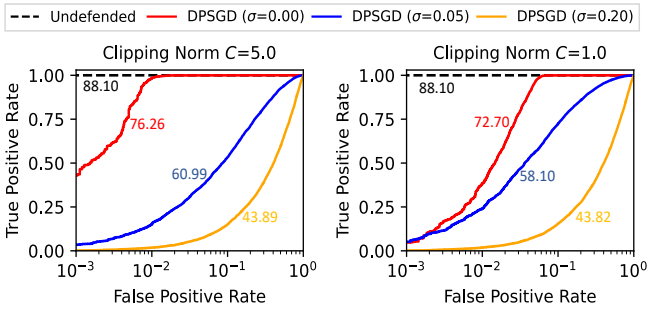

Fig. 12: Evaluating DPSGD under different clipping norms (C) and noise multipliers (σ). The numbers along each curve give the model accuracy. Overall, DPSGD is an effective defense against our attack, though it also incurs high accuracy loss. The more the noise injected, the stronger is the privacy protection, and the lower is the model accuracy.

Evade-able defenses. For those defenses that consist of their own defensive loss terms (e.g., the KL loss for soft label training in [69], [81], [26]), the adversary can create a malicious loss term for the membership-encoding samples and optimize the model on both losses. Here, our goal is to *evade* the defense and produce compromised models that can exhibit strong privacy under the standard MI evaluation. This would mislead the users into believing that their models are "private", yet in reality the models still allow the adversary to perform accurate MIA. It can conceal our attack within a seemingly private model, and render the attack even more insidious.

Non evade-able defenses. On the other hand, there exist other defenses that cannot be evaded by the malicious loss (e.g., DPSGD [13] that performs gradient perturbation). In this case, we evaluate how these defenses can reduce the privacy leakage by our attack.

We next discuss our evaluation on different defenses.

Provable defense. We consider DPSGD, a principled defense based on differential privacy [30], [13]. It bounds the influence that any sample can have on the model via performing clipping and noise injection to the gradients derived from all loss terms (hence it cannot be evaded by our attack). We evaluate different clipping norms $C \in \{1, 5\}$ and noise multipliers $\sigma \in \{0.0, 0.05, 0.2\}$, and Fig. 12 reports the results.

Using a tight clipping norm without injecting any noise (the red curve on the right of Fig. 12), DPSGD reduces the attack TPR@0.1% FPR to only 4.74%. However, it also causes a high accuracy drop of 14%. We also find that under DPSGD, our attack causes additional accuracy loss (average 3.4%), compared with the models trained without poisoning. Overall, injecting more noise further improves the privacy, but also incurs higher accuracy loss.

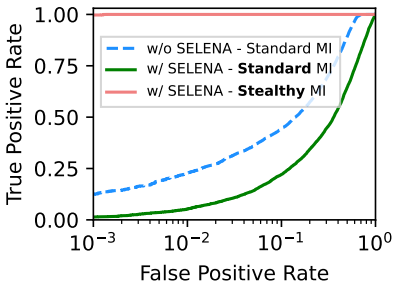

Fig. 13: Under the standard MI protocol, the model trained with the SELENA defense exhibits strong privacy protection and reduces the attack TPR@0.1% FPR from 12.46% (blue dashed line) to 1.72% (green solid line). However, our attack can be constructed to evade the defense and maintain high MI success (red line).

The trade off between privacy protection and model accuracy has been a longstanding challenge, and there are many defenses that aim to preserve high model accuracy while providing strong empirical privacy protection. We next discuss our evaluation on several such defenses.

Soft-label based defense (*evade-able*). We consider the state-of-the-art SELENA defense (USENIX Security'22) [81], based on knowledge distillation. We explain it next.

First, SELENA partitions the training set into different subsets, each of which is used to train a teacher model. For each subset, there exists another set of remaining samples that are not used to train the corresponding teacher model, and these samples are viewed as the reference non-member samples. Then, the teacher models make predictions on their respective reference samples, the outputs of which are aggregated as the privacy-preserving soft labels to be learned by the student model. In essence, the student model is trained to predict the sensitive training samples as if they are the non-member samples, which can mitigate the model's overfitting.

We first explain how we construct our attack while incorporating the SELENA defense into the training code. The privacy protection in SELENA relies on a set of privacy-preserving soft labels for performing knowledge distillation. However, they are only applied to the original training samples to derive a *defensive* loss term, and our attack can create another *malicious* loss value to evade the defense, by using both losses to optimize the model. As the adversary can use arbitrary labels for the membership-encoding samples, he/she can generate soft labels whose top-1 classes have 99% probability, for the membership-encoding samples. This produces a malicious loss that significantly facilitates the model's memorization on the synthetic samples and contributes to the high attack success.

Fig. 13 shows the results. Under the standard MI protocol, the model exhibits strong protection (offered by the SELENA defense) and reduces the attack TPR from 12.46% to only 1.72%, with a small accuracy drop of 3.35% (similar accuracy

drop on the model trained without our attack). Under the stealthy MI protocol, however, the adversary can still achieve a very high 99.73% TPR.

In the above, we demonstrate that *the proposed attack can be constructed to produce a poisoned model that conveys a sense of "strong" privacy under the standard MI protocol, yet it still secretly leaks the membership information to the adversary*. This can mislead the users into believing that the compromised models they have are "private", and render our attack even more insidious. For the adversary, he/she can provide the privacy-preserving training algorithm as an option in the training code. The users can decide to train the model with the standard (without defense) or defensive (with defense) loss term - in *either* case, our attack can continue to inflict significant privacy leakage via the compromised loss.

The use of soft labels in SELENA is a common approach and is also employed in other related defenses such as DMP [70] and Label Smoothing [80]. Thus, the success of our attack in evading the SELENA defense also has repercussions on these defenses, e.g., the adversary can provide these techniques as different training options in the training code for the users to choose from.

Defense based on adding training constraint (*evadeable*). This class of defense adds optimization constraints during training to regularize the model's behavior and reduce its privacy risk [51], [26], [59]. We evaluate a representative technique based on regularizing the output distributions [51], and show that it can be similarly evaded by optimizing the defensive and malicious loss term together (Appendix B1).

Output perturbation. This line of defense performs output obfuscation on the trained model [46], [26] and hence our attack cannot evade them. We evaluate two representative defenses [46], [26], and find that our attack still achieves high success despite the obfuscated outputs (Appendix B2).

Generic regularization. Generic techniques such as early stopping [92], dropout [77] are helpful in mitigating model overfitting and reducing privacy risk. However, they generally suffer from the trade off between privacy protection and model accuracy [67], [69], and we validate this in Appendix B3.

VI. Discussion

We first evaluate our attack under the normalization-layer-free setting (Section VI-A), and then perform a comprehensive study to understand the artifacts incurred by our attack (Section VI-B). Section VI-C presents an attack countermeasure and Section VI-D discusses the limitations of our work.

A. Is Normalization Layer Indispensable?

The proposed architectural modification to include a secondary normalization layer plays a key role in our attack. While the norm layer is a common building block in many state-of-the-art deep learning models [37], [41], [96], [40], [90], there are models that do not include the norm layer [18], [19]. This section analyzes our attack under such a normalization-free model.

We use a simpler approach by removing all the norm layers in the WideResNet model. Without the norm layer to shift and scale the inputs, we switch to using a larger mean of 0.3 and standard deviation of 1.5 for generating diverse membership-encoding samples (otherwise they cannot be memorized due to the low variance across samples). In this setting, our attack performance is not as high, though it still increases the TPR@0.1% FPR from 9.96% to 48.29% (a 4.8x increase) with 0.9% accuracy drop. We leave the improvement under the normalization-free setting to future work.

B. Discussion on Attack Artifacts

In this section, we first discuss the different artifacts incurred by our attack, and then present several alternative strategies to mitigate them.

① Additional forward passes. Our attack creates membership-encoding samples during training. They are used to create the malicious loss value, but also increase the number of forward passes at each training step. Therefore, we present an alternate method to configure our attack without increasing the number of forward passes.

The idea is to randomly *replace* a subset of training samples with their membership-encoding samples, while keeping the number of forward passes the same at each training step (e.g., 70% for training samples, and 30% for synthetic samples). The rationale is that both types of samples do not need to appear in every training step to be learned/memorized by the model, and thus we can steal the membership without increasing the number of forward passes, which also has the benefit of reducing the attack overhead (see ② next).

We evaluate our attack by replacing different portions of training samples with membership-encoding samples at each training step (10%, 30%, 50% and 70%). 10% replacement can largely preserve the model accuracy (0.65% drop), but there are limited number of membership-encoding samples at each step, which leads to a slightly lower TPR of 80%. Increasing the ratio to 30% can boost the TPR to 99.82%, at a slightly higher accuracy drop of 1.1%. Using a larger replacement ratio has negligible benefit in privacy leakage and incurs slightly higher accuracy loss (as fewer training samples are used at each training step): 50% replacement ratio has a 1.5% accuracy drop and 70% has 3.44%[3].

② Increased model complexity and overhead. The inclusion of a secondary norm layer increases the model parameters, but only to a small margin (average 0.15%). We also measure the runtime overhead by our attack (with two Nvidia V100 GPUs). The attack increases the average training time (from five repetitions) from 85.4 mins to 165 mins (93% increase). However, using the random replacement strategy introduced earlier (with a 30% replacement ratio) can reduce the runtime to 108.2 mins, which amounts to a mere 26.7% increase. Lastly, our attack also increases the average inference overhead from 7.51 ms to 7.87 ms (4.8% higher).

Despite the increased overhead, we remark that they do not make the attack easy to detect. This is because the runtime is also affected by several other factors (e.g., system environment, hardware configuration), and hence it is challenging to obtain stable runtime baselines for comparison.

[3]The replacement is done randomly and thus all training samples will still be seen by the model during training to obtain high prediction accuracy.

③ The secondary normalization layer leaves an artifact in the model's computational graph. However, to the best of our knowledge, the *existing* use of additional normalization layer in ML models is commonly intended for *benign* purposes (such as for adversarial training [88], [89], or performance improvement under imbalanced classification [95]). In contrast, our work is the *first* of its type to exploit the additional norm layer from an *adversarial* perspective to facilitate MIAs. Therefore, it is highly challenging for non-expert target users to determine that the extra norm layer was included for a malicious intent.

For completeness, we also investigate other alternative attack strategies that do *not* entail architectural modification (details in Appendix C3). This can be adopted by the adversary to eliminate the artifact of the secondary norm layer while still causing major privacy damage, though they do come with the cost of reduced attack performance (Appendix C3).

Summary. We comprehensively discussed the artifacts incurred by our attack, and also explained carefully why these artifacts do not make our attack easy to detect. In addition, we also presented several alternative methods that can greatly mitigate the attack artifacts while still inflicting considerable privacy leakage.

Overall, our attack represents a new class of MIA with several noteworthy properties, including: (1) high MI success against all training samples (average $>$99% attack TPR@0.1% FPR), (2) no reliance on shadow model calibration, and more importantly, (3) incurring negligible accuracy drop, while (4) being able to disguise the amplified privacy leakage under common membership privacy auditing. These together represent a significant advancement over existing poisoning-based MIAs [75], [25], [82]. We leave further improvement in covering the remaining attack traces to follow-up studies.

C. Attack Countermeasure.

Our evaluation in Section V-F3 on various MIAs defenses illustrates the challenges in mitigating the proposed attack. However, our attack relies on the exact knowledge of the target sample, for generating the unique random seed to reconstruct the membership-encoding sample. Hence, one countermeasure is to slightly modify the target sample so that the adversary cannot generate the same random seed to reconstruct the secret sample. The modifications can manifest in different forms, and thus they are hard to predict by the adversary.

With that said, the exact knowledge of target sample is still a common assumption in existing practice of MIAs [72], [93], [21], [39]. Hence, our attack still poses significant privacy threat, and a systematic amendment of the existing MI defenses to handle our attack is another avenue for future work.

D. Limitation

The major limitation of our work concerns the feasibility of mounting code poisoning attacks in practice. While code poisoning attacks have been shown to be feasible in real-world ML codebase [9], [78], [11], we have not realized our attack in the open world, and neither have any prior code-poisoning attack studies [15], [73], [75], [54], to the best of our knowledge. This is a limitation of this class of studies.

Nevertheless, launching the attack in the open world would require strong controls to prevent any actual harm to users of the ML infrastructure. Any controlled attack that seeks to amplify the privacy leakage of production ML models should carefully consider how to address the related ethical concerns, an open question that requires further research.

Finally, while our attack is yet to be realized in practice, we hope the comprehensive proof of concept of the proposed attack can bring awareness to the risk of code poisoning in third-party ML codebase, which becomes highly relevant given that integrating external code repositories in the development of ML model is becoming a growing practice. Thus, our work also calls for future efforts on ML code security and we outline several such directions next.

VII. Conclusion and Future Work

This work introduces a new form of membership inference attack against deep learning models, based on poisoning two opaque and difficult-to-test modules in the model-training code: the loss-value computation and model structure. The training code can be used by the victim users in a trusted environment to produce compromised models that can operate faithfully on the main task with competitive performance, while still secretly leaking the membership information of all the training samples to the black-box adversary.

Our work illustrates how the massive learning capacity of modern deep learning models can be exploited by the adversary to amplify membership privacy leakage in a secret manner. The amplified privacy leakage inflicted by the attack can remain *unnoticeable* under common privacy auditing methods, and a deliberate adversary can go even further to disguise the attack by tricking the corrupted model to convey a *false* sense of strong privacy and mislead the users. From this, we outline three directions to be explored in future studies.

(1) Rethinking the current membership privacy auditing practice. Existing auditing practice does not account for code inspection, which is a necessary step in exposing the privacy leakage inflicted by our attack. Thus, an open question is should the model-training code be supplied as part of the inputs in the standard membership inference game, though identifying the malicious constructs from the complicated codebase itself can become another barrier? In addition, a direct approach to thwart our current attack is to slightly modify the target sample. Thus, developing a standardized approach to enact this can be another avenue for future study.

(2) Extending our attack to other settings. There are several directions that can be explored to extend our attack, including attack extension to generative models [36], [24], to other domains such as natural language processing [70], [58], and improving the attack performance under other challenging settings (e.g., normalization-free scenario).

(3) Developing more capable defenses and code analysis tools. Our evaluation on existing defense techniques shows that the challenge in providing strong privacy protection without incurring high accuracy loss still remains. Those prior privacy defenses that can achieve a superior privacy-utility trade off under standard MI evaluation, unfortunately can be "evaded" by a deliberate adversary, and future work can study the potential of more capable defenses to withstand such attack.

Another related direction is to develop automated code inspection tools to analyze the irregular and potentially malicious logic in the ML codebase. There are a number of code integrity checking tools for traditional software [20], [22]. Future work can study whether they can be adapted to counter code poisoning attacks in ML codebases, where the key challenge is that a similarly irregular code logic can be intended for either a benign or malicious purpose. For example, a trusted computational graph [15] can be used to detect whether the malicious code has caused the model to deviate from the expected computational graph during training, such as creating an additional forward pass. However, the adversary can still replace a subset of training samples with the membership-encoding samples to mount the attack without causing an extra forward pass (Section VI-B). Software signing [5], [8], [62], where the maintainers digitally sign their released codebases to prevent unauthorized code manipulation, is also a promising direction. However, in modern ML ecosystems where many codebases are maintained by multiple developers and undergo frequent iterations, the potential usability concerns [62], [68] are another factor to be considered.

ACKNOWLEDGMENT

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), a grant from the National Research Councile of Canada (NRC), and a Four Year Fellowship from the University of British Columbia.

REFERENCES

[1] https://github.com.

[2] https://about.gitlab.com.

[3] https://huggingface.co.

[4] "Adversarial threat landscape for artificial-intelligence systems." https://atlas.mitre.org.

[5] "Cis software supply chain security guide." Center for Internet Security, https://www.cisecurity.org/insights/white-papers/cis-software-supply-chain-security-guide.

[6] "Ml privacy meter," https://github.com/privacytrustlab/ml_privacy_meter.

[7] "Nist ai risk management framework." https://airc.nist.gov/AI_RMF_Knowledge_Base/AI_RMF.

[8] "Open source software (oss) secure supply chain (ssc) framework simplified requirements." Microsoft, https://github.com/microsoft/oss-ssc-framework/blob/main/specification/framework.md.

[9] "Pytorch dependency poisoned with malicious code," https://www.theregister.com/2023/01/04/pypi_pytorch_dependency_attack/.

[10] "Pytorch implementation of wideresnet architecture," https://github.com/meliketoy/wide-resnet.pytorch.

[11] "Tensorflow ci/cd flaw exposed supply chain to poisoning attacks," https://thehackernews.com/2024/01/tensorflow-cicd-flaw-exposed-supply.html.

[12] "Tensorflow privacy | responsible ai toolkit," https://www.tensorflow.org/responsible_ai/privacy/guide.

[13] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[14] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[15] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1505–1521.

[16] M. Bertran, S. Tang, A. Roth, M. Kearns, J. H. Morgenstern, and S. Z. Wu, "Scalable membership inference attacks via quantile regression," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[17] F. Boenisch, V. Battis, N. Buchmann, and M. Poikela, ""i never thought about securing my machine learning systems": A study of security and privacy awareness of machine learning practitioners," in *Proceedings of Mensch und Computer 2021*, 2021, pp. 520–546.

[18] A. Brock, S. De, and S. L. Smith, "Characterizing signal propagation to close the performance gap in unnormalized resnets," *arXiv preprint arXiv:2101.08692*, 2021.

[19] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1059–1071.

[20] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer, "Control-flow integrity: Precision, security, and performance," *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, pp. 1–33, 2017.

[21] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership inference attacks from first principles," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1897–1914.

[22] M. Castro, M. Costa, and T. Harris, "Securing software by enforcing data-flow integrity," in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 147–160.

[23] H. Chaudhari, J. Abascal, A. Oprea, M. Jagielski, F. Tramèr, and J. Ullman, "Snap: Efficient extraction of private properties with poisoning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 1935–1952.

[24] D. Chen, N. Yu, Y. Zhang, and M. Fritz, "Gan-leaks: A taxonomy of membership inference attacks against generative models," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 343–362.

[25] Y. Chen, C. Shen, Y. Shen, C. Wang, and Y. Zhang, "Amplifying membership exposure via data poisoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 29 830–29 844, 2022.

[26] Z. Chen and K. Pattabiraman, "Overconfidence is a dangerous thing: Mitigating membership inference attacks by enforcing less confident prediction," *arXiv preprint arXiv:2307.01610*, 2023.

[27] C. A. Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot, "Label-only membership inference attacks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1964–1974.

[28] J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multiobjective optimization," *Comptes Rendus Mathematique*, vol. 350, no. 5-6, pp. 313–318, 2012.

[29] J. Dressel and H. Farid, "The accuracy, fairness, and limits of predicting recidivism," *Science advances*, vol. 4, no. 1, p. eaao5580, 2018.

[30] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 2006, pp. 265–284.

[31] V. Feldman, "Does learning require memorization? a short tale about a long tail," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020, pp. 954–959.

[32] V. Feldman and C. Zhang, "What neural networks memorize and why: Discovering the long tail via influence estimation," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2881–2891, 2020.

[33] L. Fowl, J. Geiping, S. Reich, Y. Wen, W. Czaja, M. Goldblum, and T. Goldstein, "Decepticons: Corrupted transformers breach privacy in federated learning for language models," *arXiv preprint arXiv:2201.12675*, 2022.

[34] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[35] J. J. Hathaliya and S. Tanwar, "An exhaustive survey on security and privacy issues in healthcare 4.0," *Computer Communications*, vol. 153, pp. 311–335, 2020.

[36] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro, "Logan: Membership inference attacks against generative models," *arXiv preprint arXiv:1705.07663*, 2017.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[39] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, "Membership inference attacks on machine learning: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.

[40] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[41] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[42] B. Hui, Y. Yang, H. Yuan, P. Burlina, N. Z. Gong, and Y. Cao, "Practical blind membership inference attack via differential comparisons," in *ISOC Network and Distributed System Security Symposium (NDSS)*, 2021.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[44] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *USENIX Security Symposium*, 2019.

[45] B. Jayaraman, L. Wang, K. Knipmeyer, Q. Gu, and D. Evans, "Revisiting membership inference under realistic assumptions," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 2, 2021.

[46] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "Memguard: Defending against black-box membership inference attacks via adversarial examples," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 259–274.

[47] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[49] R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia, "Adversarial machine learning-industry perspectives," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020, pp. 69–75.

[50] K. Leino and M. Fredrikson, "Stolen memories: Leveraging model memorization for calibrated white-box membership inference," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1605–1622.

[51] J. Li, N. Li, and B. Ribeiro, "Membership inference attacks and defenses in classification models," in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, 2021, pp. 5–16.

[52] Z. Li and Y. Zhang, "Membership leakage in label-only exposures," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 880–895.

[53] Y. Liu, Z. Zhao, M. Backes, and Y. Zhang, "Membership inference attacks by exploiting loss trajectory," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2085–2098.

[54] Z. Liu, F. Li, Z. Li, and B. Luo, "Loneneuron: a highly-effective feature-domain neural trojan using invisible and polymorphic watermarks," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2129–2143.

[55] S. Mahloujifar, E. Ghosh, and M. Chase, "Property inference from poisoning," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1120–1137.

[56] M. Malekzadeh, A. Borovykh, and D. Gündüz, "Honest-but-curious nets: Sensitive attributes of private inputs can be secretly coded into the classifiers' outputs," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 825–844.

[57] J. Mink, H. Kaur, J. Schmüser, S. Fahl, and Y. Acar, ""' security is not my field, i'm a stats guy": A qualitative root cause analysis of barriers to adversarial machine learning defenses in industry," in *In 32nd USENIX Security Symposium*, 2023.

[58] F. Mireshghallah, K. Goyal, A. Uniyal, T. Berg-Kirkpatrick, and R. Shokri, "Quantifying privacy risks of masked language models using membership inference attacks," *arXiv preprint arXiv:2203.03929*, 2022.

[59] M. Nasr, R. Shokri, and A. Houmansadr, "Machine learning with membership privacy using adversarial regularization," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 634–646.

[60] ——, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[61] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[62] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2353–2367.

[63] D. Nguyen, S. Gupta, T. Nguyen, S. Rana, P. Nguyen, T. Tran, K. Le, S. Ryan, and S. Venkatesh, "Knowledge distillation with distribution mismatch," in *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21*. Springer, 2021, pp. 250–265.

[64] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics, 2019.

[65] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[66] N. Ponomareva, H. Hazimeh, A. Kurakin, Z. Xu, C. Denison, H. B. McMahan, S. Vassilvitskii, S. Chien, and A. Thakurta, "How to dp-fy ml: A practical guide to machine learning with differential privacy," *arXiv preprint arXiv:2303.00654*, 2023.

[67] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," *arXiv preprint arXiv:1806.01246*, 2018.

[68] T. R. Schorlemmer, K. G. Kalu, L. Chigges, K. M. Ko, E. A.-M. A. Isghair, S. Baghi, S. Torres-Arias, and J. C. Davis, "Signing in four public software package registries: Quantity, quality, and influencing factors," *arXiv preprint arXiv:2401.14635*, 2024.

[69] V. Shejwalkar and A. Houmansadr, "Membership privacy for machine learning models through knowledge transfer," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, pp. 9549–9557, May 2021.

[70] V. Shejwalkar, H. A. Inan, A. Houmansadr, and R. Sim, "Membership inference attacks against nlp classification models," in *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

[71] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi, "Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis," *IEEE journal of biomedical and health informatics*, vol. 22, no. 5, pp. 1589–1604, 2017.

[72] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

[73] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, 2017, pp. 587–601.

[74] C. Song and V. Shmatikov, "Overlearning reveals sensitive attributes," in *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[75] C. Song and R. Shokri, "Robust membership encoding: Inference attacks and copyright protection for deep learning," *arXiv preprint arXiv:1909.12982*, 2019.

[76] L. Song and P. Mittal, "Systematic evaluation of privacy risks of machine learning models," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[77] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[78] J. Stawinski, "Playing with fire - how we executed a critical supply chain attack on pytorch," https://johnstawinski.com/2024/01/11/playing-with-fire-how-we-executed-a-critical-supply-chain-attack-on-pytorch/comment-page-1/.

[79] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[80] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[81] X. Tang, S. Mahloujifar, L. Song, V. Shejwalkar, M. Nasr, A. Houmansadr, and P. Mittal, "Mitigating membership inference attacks by {Self-Distillation} through a novel ensemble architecture," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1433–1450.

[82] F. Tramèr, R. Shokri, A. San Joaquin, H. Le, M. Jagielski, S. Hong, and N. Carlini, "Truth serum: Poisoning machine learning models to reveal their secrets," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 2779–2792.

[83] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[84] Y.-X. Wang, B. Balle, and S. P. Kasiviswanathan, "Subsampled rényi differential privacy and analytical moments accountant," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1226–1235.

[85] Y. Wen, A. Bansal, H. Kazemi, E. Borgnia, M. Goldblum, J. Geiping, and T. Goldstein, "Canary in a coalmine: Better membership inference with ensembled adversarial queries," *arXiv preprint arXiv:2210.10750*, 2022.

[86] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[87] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[88] C. Xie, M. Tan, B. Gong, J. Wang, A. L. Yuille, and Q. V. Le, "Adversarial examples improve image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 819–828.

[89] C. Xie and A. Yuille, "Intriguing properties of adversarial training at scale," in *International Conference on Learning Representations*, 2019.

[90] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[91] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[92] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[93] J. Ye, A. Maddi, S. K. Murakonda, V. Bindschaedler, and R. Shokri, "Enhanced membership inference attacks against machine learning models," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3093–3106.

[94] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 268–282.

[95] S. Zada, I. Benou, and M. Irani, "Pure noise to the rescue of insufficient data: Improving imbalanced classification by training on random noise images," in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 817–25 833.

[96] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[97] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *International Conference on Learning Representations*, 2017.

[98] ——, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[99] Y. Zhang, G. Bai, M. A. P. Chamikara, M. Ma, L. Shen, J. Wang, S. Nepal, M. Xue, L. Wang, and J. Liu, "Agrevader: Poisoning membership inference against byzantine-robust federated learning," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2371–2382.

APPENDIX

A. Evaluation on Different Training Sizes

We evaluate 8 different sizes (2,500 to 25,000). On average, our attack achieves >99% TPR@0.1% FPR with 0.43% accuracy drop - the detailed results can be referred to the paper's extended version [link].

B. Additional Results on Defense Evaluation

This section reports the results on three different types of defenses based on: (1) adding training constraint, (2) output perturbation, and (3) generic regularization techniques.

We summarize the key findings below, and the detailed results can be referred to the paper's extended version [link].

1) ***Defense based on adding training constraint****:* We evaluate the defensive regularization based on Maximum Mean Discrepancy (MMD) in Li et al. [51]. We evaluate this defense using different five different parameters from $[1, 7]$ to vary the regularization strength (larger values cause excessive regularization, and the model cannot be trained to obtain good accuracy). Overall, we find that our attack still achieves consistently high success in all settings (>99% TPR@0.1% FPR and average <1% accuracy drop). This is because the regularization term is applied only to the outputs on the original training samples, and our attack can create the malicious loss term from the membership-encoding samples to maintain its success (similar to what we did in Section V-F3).

There are other related defenses with different regularization forms such as the min-max game in AdvReg [59], and our attack can be constructed in a similar way to evade them.

2) ***Defense based on output obfuscation*** *:* These defenses seek to obfuscate the output vector to reduce privacy leakage, and we evaluate MemGuard, which perturbs the output vector to confuse a shadow MI classifier [46]; and HAMP[4], which replaces the output vector with a randomized vector (from the output on a randomly created sample) while preserving only the relative ordering within the vector, i.e., only the label-related information are preserved [26].

[4]HAMP consists of soft label training, adding training constraint, and output obfuscation. Our earlier evaluations show that the first two components can be evaded by our attack (as in Section V-F3 and Appendix B1), and hence we evaluate the last component (output obfuscation defense) against our attack.

For MemGuard, we first validate that it is able to reduce the attack accuracy of the shadow MI classifier from ∼99% to 50%. Yet prior work find that this approach of MemGuard still provides very limited privacy protection [81], [76], [69]; and similarly our attack still achieves 100% TPR@0.1% FPR.

For HAMP, we use the neural network (NN) based attack by Nasr et al. [60] (which we find to be effective against such randomization-based defenses). We follow prior work [59], [81], [69], [26] to use the first half of the members and non-members to train the attack model, and perform evaluation on the remaining half. In this case, our attack still achieves very high MI success, with 90.98% TPR@0.1% FPR.

3) ***Generic regularization techniques****:* We evaluate two common techniques: dropout [77] and early stopping [92]. Both defenses still suffer from the trade off between privacy and utility, e.g., aggressive regularization can degrade the attack success, but at the cost of accuracy drop (vice versa).

C. Ablation Study

Our attack consists of the following components: (1) mean and standard deviation for generating membership-encoding samples, (2) the labels of these samples, and (3) the inclusion of secondary normalization layer. We next conduct an ablation study for each of these components.

1) ***Impact of different mean and standard deviation values for the membership-encoding samples****:* To select the mean and standard deviation for the membership-encoding samples, we analyze the sample statistics of $2,000$ shadow samples. Fig. 14 shows a visual representation of the same. Based on this, we select a wide range of values that are different from those of the shadow samples to specify the membership-encoding samples.

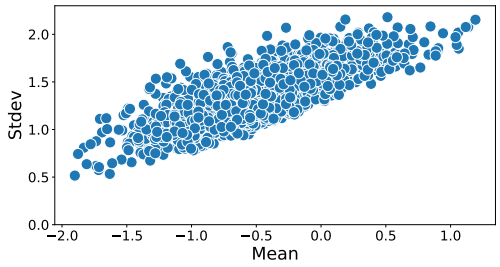

Fig. 14: Visualizing the sample mean and standard deviation of a set of shadow samples, based on which the adversary can specify the values to generate the membership-encoding samples (e.g., mean of 0. and stdev of 0.1).

Specifically, we consider different means from (0.0, 0.2, 0.5, -0.2, -0.5), and standard deviations from (0.1, 0.2, 0.4, 0.7), totaling 20 different configurations. In determining whether a sample follows the adversary-specified sample mean and standard deviation (stdev), we empirically add a small absolute offset of 0.1, as the sample's mean and stdev may not precisely match the specified values (e.g., a sample specified with 0.1 stdev may have 0.105 stdev). The detailed results and an illustration of these samples can be referred to the paper's extended version [link].

Overall, our attack achieves high attack TPR (average 99.98% TPR) and low accuracy drop (average 0.63%) across different mean and standard deviation values.

Additional analysis. We now discuss a hypothetical scenario where the means and standard deviations of the membership-encoding samples coincide with the training samples', i.e., some of the training samples will be routed to the secondary normalization layer, and we study how this affects our attack.

We first analyze the sample mean and standard deviation of all training samples, and select the five most frequent pairs of (mean, standard deviation): (-0.40, 1.30), (-0.50, 1.25), (-0.45, 1.20), (-0.55, 1.20) and (-0.45, 1.40), which encompasses 6.9% to 7.3% of the training samples. These are the highest percentages because we find that the training samples have diverse sample means and standard deviations and they do not concentrate on a specific region (e.g., the densest region contains <8% of the samples).

Using the above parameters in configuring our attack leads to a slight drop of attack performance, and the attack TPR@0.1% FPR is reduced to 95.12%∼99.69% (average 96.98%), but the accuracy remains similar (-0.6% Vs. -0.58%). Overall, we find the degradation is only marginal, and our attack still has very high performance.

Moreover, the above scenario can be prevented by the adversary using a set of shadow samples to guide the selection of mean and standard deviation, such as those evaluated earlier.

2) ***Impact of the labels for the membership-encoding samples*** *:* In specifying the label of the membership-encoding sample (x^*), we mentioned earlier in the attack setup (Section V-A) that its label y^* can be set to be a random label as long as the adversary knows how to recover it. To validate this, we use the hash value from the target sample (x) as the random seed to generate a random label for y^* during training. The adversary can reconstruct it to perform the stealthy MI at inference time.

We train a model under this approach, and find that our attack achieves similar success as before, with 100% TPR@0.1% FPR with 0.1% accuracy drop. This is because the membership-encoding samples are designed to bear no discernible features to any of the class labels, and thus the attack can succeed despite the choice of label.

3) ***Alternative attack design****:* Our main attack design consists of a secondary normalization layer to separately process the training and membership-encoding samples. The earlier evaluation in Section V-F2 also validates the effectiveness of this solution in overcoming the distribution mismatch problem, and it leads to the greatly improved attack success and much lower accuracy drop.

We now discuss two alternative attack strategies, which, compared with the main approach, do not require architectural modification, but are somewhat less effective.

In particular, instead of using different normalization layers to process the training and membership-encoding samples, our idea is to configure the scaling coefficients to balance the losses on these two types of samples. We instantiate this idea into two approaches.

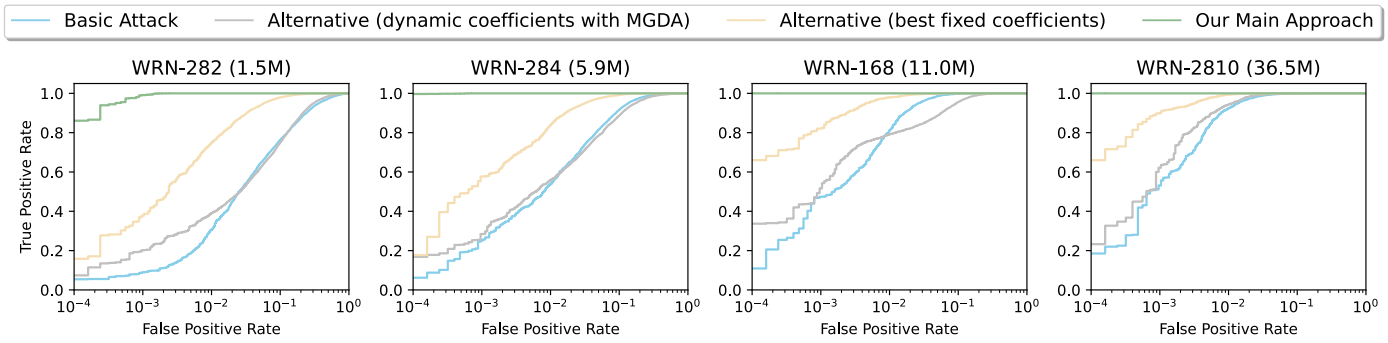

Fig. 15: Comparing the attack performance under different attack strategies. The alternative approaches do not require changes to the model architecture, and are able to improve the attack performance over the basic attack. On average, the attack TPR@0.01% FPR on different attack methods are: 10.28% (basic attack), 20.31% (MGDA-based alternative method), 41.40% (fixed-coefficients-based alternative method), and 96.44% (the main approach).

The first approach is inspired by Bagdasaryan et al. [15], and we use the Multiple Gradient Descent Algorithm (MGDA) [28] to find the scaling coefficients that can minimize the losses on the training and membership-encoding samples:

$$\min_{\alpha,\beta}\{\| \alpha\nabla\ell_{train} + \beta\nabla\ell_{synthetic} \|_2^2 \mid \alpha,\beta \geq 0\}, \quad (2)$$

where $\nabla\ell$ represents the gradients associated with the training and synthetic samples. We follow Bagdasaryan et al. [15] to pass the losses and gradients to MGDA, and compute the scaling coefficients α, β. Note that the basic attack (in Section IV-D) that directly trains the model on the training and membership-encoding samples can be viewed as using the same coefficients in Equation 2 (i.e., both sets of samples have equal weight).

In addition to the dynamic coefficients by MGDA, we consider a second alternative of using fixed coefficients by experimenting with different values, and then selecting the one with the highest attack performance.

Evaluation setup. Our attack increases the membership leakage by inducing the model to memorize a set of membership-encoding samples, and the memorization effect is related to the model's capacity. We therefore conduct evaluation across different models with various capacities (from 1.5 million to over 36 million parameters).

For the MGDA-based alternative approach, we use the original implementation from Bagdasaryan et al. [15]. For the other approach that uses fixed coefficients, we fix α as 1, and evaluate $\beta \in \{0.3, 0.6, 2, 3, 5, 10\}$ (using larger β value yields poorer attack performance and higher accuracy drop).

Results. We compare the two alternative strategies with our main approach: Fig. 15 compares the attack performance, and Fig. 16 reports the accuracy drop by different approaches.

In terms of attack performance (Fig. 15), the two alternative approaches are able to improve the attack performance over the basic attack. The MGDA-based approach that uses dynamic coefficients improves the average attack TPR@0.1% FPR from 33.65% to 40.62%, and attack TPR@0.01% from 10.28% to 20.31% The other approach that uses the best fixed coefficients is able to yield better attack performance : on average, it

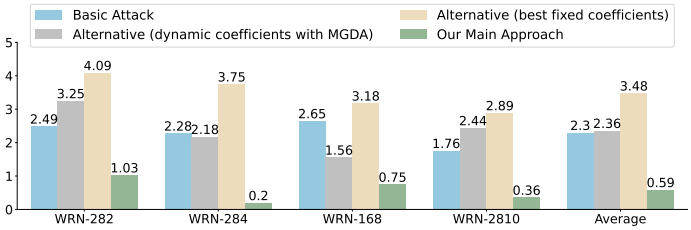

Fig. 16: Accuracy drop incurred by different attack strategies.

achieves 66.86% TPR@0.1% FPR, and 41.40% TPR@0.01% FPR. Meanwhile, we observe that the attack performance improves as the model capacity grows, which is understandable since larger models tend to have more capacity to memorize the membership-encoding samples. With that said, the two alternate methods' performance are still lagging behind the main approach's. On average, the TPR@0.01% FPR across different attacks are: 10.28% (basic attack); 20.31% (MGDA-based alternative method); 41.40% (fixed-coefficients-based alternative method); and 96.44% (our main approach).

Further, both alternative approaches increase the membership leakage at the cost of slightly higher accuracy drop (Fig. 16). The average accuracy drop by the MGDA-based approach is 2.36%, and 3.48% by the fixed-coefficient-based approach. The former has lower accuracy drop as it aims to balance the goal of low accuracy drop and high privacy leakage; whereas the latter seeks to maximize the attack performance (which comes at the cost of higher accuracy degradation, as shown). Compared with these two approaches, the main attack strategy incurs a lower accuracy drop of 0.59%.

Summary. Our study shows that the proposed attack can be mounted in several strategies with different properties. We introduce two alternative approaches that work by configuring the coefficients to balance the losses on the training and membership-encoding samples. They have the benefit of not requiring modification to the model's structure, and still being able to amplify the membership leakage to a considerable extent (Fig. 15), though they do exhibit less profound performance compared with the main attack approach that consists of using a secondary normalization layer. Together, these different attack strategies represent a comprehensive list of privacy threats that can be posed by the adversary.

APPENDIX

A. Description & Requirements

1) How to access: The artifact package can be accessed at https://doi.org/10.5281/zenodo.13709528.

2) Hardware dependencies: Commodity GPUs (e.g., we used Nvidia RTX 3090, A5000, V100 GPUs in our evaluation).

3) Software dependencies: PyTorch, torchvision, pandas scikit-learn scipy matplotlib numpy imagehash.

4) Benchmarks: (1) *Datasets*: CIFAR10, GTSRB, SVHN, MedMnist and CIFAR100. (2) *Models*: WideResNet (with varying depth and width), DenseNet, SENet, and RexNext.

B. Artifact Installation & Configuration

Install the software dependencies listed in A-3 above (more details available in the README file in the artifact package).

The dataset used for the artifact evaluation will be automatically downloaded when running the experiments.

C. Major Claims

- (C1): Our attack empowers accurate membership inference (MI) against all training samples: average $> 99\%$ true-positive-rate (TPR)@0.1% false-positive-rate (FPR). This is proven by the experiment (E1) whose results are reported in Fig. 8 (under different *datasets*), Table I (under different *models*) and Table-III (under different *training-set sizes*).
- (C2): In our attack, the stolen membership can be inferred without the expensive shadow-model calibration. This is proven by the experiment (E1) whose results are reported in Fig. 10.
- (C3): The poisoned models exhibit comparable *accuracy* as the non-poisoned models (average $< 1\%$ difference). This is proven by the experiment (E1 and E2) whose results are reported in Fig. 9.
- (C4): The poisoned models exhibit comparable *privacy leakage* (under common MI evaluation methods) as the non-poisoned models (average $< 1\%$ difference). This is proven by the experiment (E2) whose results are reported in (the green and blue lines of) Fig. 8.

D. Evaluation

Reference to the key results (at the end of Section I): Our attack: (1) empowers accurate MI against all training samples (average $> 99\%$ TPR@0.1% FPR), and (2) the stolen membership can be inferred without the expensive shadow-model calibration.

Furthermore, the poisoned models exhibit comparable (3) accuracy and (4) privacy leakage under common MI evaluation methods, as the non-poisoned models (average $< 1\%$ difference on both regards).

NOTE. Due to the time limit of the artifact evaluation, we have made the following arrangements.

① In the paper, Claims C1, C2 and C3 are evaluated under 5 benchmark datasets, 10 models, 8 training-set sizes. In the artifact evaluation, the experiments are scaled down to 1 dataset (CIFAR10), 2 models (WideResNet-28-10 and SENet) and 2 training-set sizes (12,500 and 5,000). With that said, we have observed a consistent trend on both the full-scale and scaled-down experiments.

② Validating Claim C4 requires executing the likelihood-ratio attack (LiRA) and training multiple shadow models, which is infeasible due to the time constraint.

Therefore, we have designed a modified version of the experiment that can enable a quick reproduction of the LiRA experiments *without* training shadow models. Specifically, the shadow models are used to make predictions on the shadow samples, and we have provided the *pre-computed* outputs from the shadow models used in our experiments. These data can be used to directly perform MI on the target model, and validate C4 (details in Experiment E2 later).

*1) **Experiment (E1):*** E1 is to validate Claim C1, C2 and C3. We validate these claims by performing experiments across multiple experimental settings (datasets, models and training-set sizes); and we provide a single script that consolidates all the experiments. Each experiment trains a model and measures its accuracy and privacy leakage.

Step 1: Train the poisoned model with our attack.

[Execution] Open a `screen` session and run `bash eval-codePoisoned.sh &> Output-eval-codePoisoned`.

[Results] Read the output file `Output-eval-codePoisoned` (e.g., Fig. 17) to validate C1 and C2, as follows.

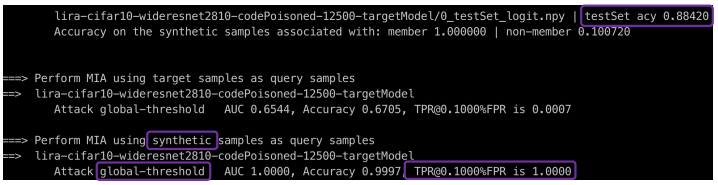

Fig. 17: Example output showing **model accuracy** and **privacy leakage** of the *poisoned* model trained under the proposed attack. The privacy leakage is exposed by performing the global-threshold-based LiRA method via querying the target model with the membership-encoding (synthetic) samples.

C1 can be validated by checking the attack TPR@0.1% FPR when using the *membership-encoding (synthetic) samples* as the query samples for MI (e.g., 100% TPR@0.1% FPR in a CIFAR10 model in the bottom-right purple box in Fig. 17).

C2 can be simultaneously validated by the above results, as they are obtained from the global-threshold-based LiRA method, which does not require shadow models.

[Remark] Fig. 17 also shows another way to perform MI by using *target samples* as query samples (which is a common practice in existing MI methods), but this has limited performance as shown. A common strategy to overcome this is to train some shadow models and calibrate the MI process - we will evaluate the shadow-model-based method in Experiment E2 when validating Claim C4.

Step 2: Train the uncorrupted model.

[Execution] Open a `screen` session and run `bash eval-uncorrupted.sh &> Output-eval-uncorrupted`.

[Results] Read the output file `Output-eval-uncorrupted`, which, when combined with `Output-eval-codePoisoned` in Step 1, can be used to measure the *accuracy difference* between the uncorrupted and poisoned model, and validate C3 (average $< 1\%$ accuracy drop).

*2) **Experiment (E2)**:* E2 is to validate Claim C3 and C4.

[Preparation] Download the pre-computed scores at this link (about 15GB data), and then unzip it under the main directory of the artifact package.

[Execution] Run `bash lira-quick-reproduce.sh &> Output-lira-quick-reproduce`. This leverages the pre-computed scores to perform LiRA on the uncorrupted and poisoned models (across all 5 datasets we evaluated). For each evaluation, the pre-computed scores are obtained from 128 shadow models.

[Results] Read the output file `Output-lira-quick-reproduce` (e.g., Fig. 18) to validate Claim C3 and C4.

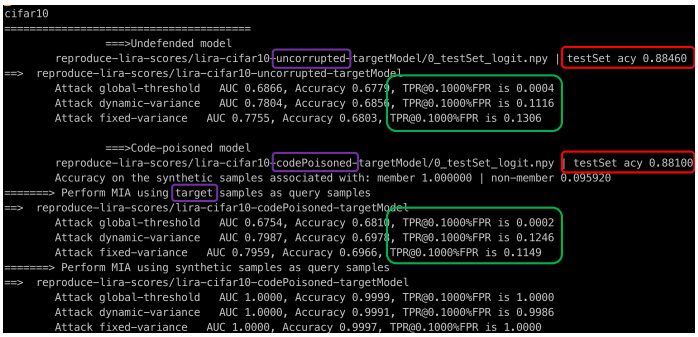

Fig. 18: Example output comparing the uncorrupted and poisoned models in terms of their **model accuracy** (*red* boxes), and **privacy leakage** (*green* boxes). In the latter, we use the three LiRA methods for MI, and they use the *target* samples as query samples as per existing MI practice. On *both* regards, the poisoned model exhibits *comparable* behaviors as its uncorrupted counterpart, which renders them highly indistinguishable from each other.

C3 can be similarly validated by comparing the accuracy difference between the uncorrupted and poisoned model (e.g., the red boxes in Fig. 18).

C4 can be validated by comparing the *highest* privacy leakage on the uncorrupted and poisoned model, when both are queried with *target samples* (as per existing MI practice).

For instance, in the green boxes in Fig. 18, the highest TPR@0.1% FPR on the uncorrupted model is 13.06% vs. 12.46% on the poisoned model. This validates that the poisoned model exhibits similar privacy leakage as the its uncorrupted counterpart.

Finally, in the bottom area of Fig. 18, when using the *synthetic* samples as query samples for MI (this applies only to the code-poisoned model), all three MI methods can be used by the adversary similarly achieve $> 99\%$ TPR@0.1% FPR.