PPA: Preference Profiling Attack
Against Federated Learning

Chunyi Zhou[†§*], Yansong Gao[†*],
Anmin Fu[†§‡], Kai Chen[§], Zhiyang Dai[†], Zhi Zhang[¶], Minhui Xue[¶], and Yuqing Zhang[∥]

[†]*School of Computer Science and Engineering, Nanjing University of Science and Technology*, China
[§]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Science*, China
[¶]*Data61, CSIRO*, Syndey, Australia
[∥]*National Computer Network Intrusion Protection Center, University of Chinese Academy of Science*, China
{zhouchunyi;yansong.gao;fuam;dzy}@njust.edu.cn; chenkai@iie.ac.cn;
{zhi.zhang; jason.xue}@data61.csiro.au; zhangyq@ucas.ac.cn
Co-first authors; [‡]*Corresponding author*

Abstract—Federated learning (FL) trains a global model across a number of decentralized users, each with a local dataset. Compared to traditional centralized learning, FL does not require direct access to local datasets and thus aims to mitigate data privacy concerns. However, data privacy leakage in FL still exists due to inference attacks, including membership inference, property inference, and data inversion.

In this work, we propose a new type of privacy inference attack, coined Preference Profiling Attack (PPA), that accurately profiles the private preferences of a local user, e.g., most liked (disliked) items from the client's online shopping and most common expressions from the user's selfies. In general, PPA can profile top-k (i.e., $k = 1, 2, 3$ and $k = 1$ in particular) preferences contingent on the local client (user)'s characteristics. Our key insight is that the gradient variation of a local user's model has a distinguishable sensitivity to the sample proportion of a given class, especially the majority (minority) class. By observing a user model's gradient sensitivity to a class, PPA can profile the sample proportion of the class in the user's local dataset, and thus *the user's preference of the class* is exposed. The inherent statistical heterogeneity of FL further facilitates PPA. We have extensively evaluated the PPA's effectiveness using four datasets (MNIST, CIFAR10, RAF-DB and Products-10K). Our results show that PPA achieves 90% and 98% top-1 attack accuracy to the MNIST and CIFAR10, respectively. More importantly, in real-world commercial scenarios of shopping (i.e., Products-10K) and social network (i.e., RAF-DB), PPA gains a top-1 attack accuracy of 78% in the former case to infer the most ordered items (i.e., as a commercial competitor), and 88% in the latter case to infer a victim user's most often facial expressions, e.g., disgusted. The top-3 attack accuracy and top-2 accuracy is up to 88% and 100% for the Products-10K and RAF-DB, respectively. We also show that PPA is insensitive to the number of FL's local users (up to 100 we tested) and local training epochs (up to 20 we tested) used by a user. Although existing countermeasures such as dropout and differential privacy protection can lower the PPA's accuracy to some extent, they unavoidably incur notable deterioration to the global model. The source code is available at https://github.com/PPAattack.

I. INTRODUCTION

As traditional centralized deep learning requires aggregating user data into one place, it can be abused to leak user data privacy and breaches the national regulations such as General Data Protection Regulation (GDPR) [37], California Privacy Rights Act (CPRA) [44], and China Data Security Law (CDSL) [10]. In contrast to centralized learning, Federated learning (FL) [31] voids the requirement by using local user model updates rather than raw user data, significantly mitigating data privacy leaks [52], [55], [63], [5]. As such, FL has become the most popular distributed machine learning technique and empowered a wide range of privacy-sensitive applications, such as smart healthcare, social network and wireless communication [59], [45], [62], [35], [42], [9].
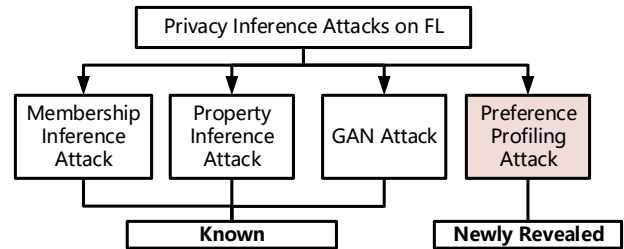

Fig. 1: Privacy inference attacks on FL.

State-of-the-art Attacks: Nonetheless, private user information in FL can still be leaked by inference attacks [33], as summarized in Figure 1. To date, there are three known privacy inference attacks. Specifically, a membership inference attack [41], [34] determines whether a particular data record is used for training a victim user's uploaded model. A property inference attack [13], [32] infers whether a target attribute exists in a victim user's local dataset. A Generative Adversarial Networks (GAN) attack [19], [50] reverse engineers images of a target label based on the model uploaded by a victim user. In privacy-sensitive applications, these inference attacks can leak private information of a target user, posing severe threats to user privacy in FL [30]. Taking the DNA determination as an example, the membership inference attack can utilize

a statistical distance measurement to determine if a known individual is in a mixture of DNA [16].

None of the aforementioned privacy inference attacks can profile local user data preference of FL, one of the most sensitive types of personal information an attacker is interested in, akin to profiling users in social networks [6], [53], [3], [15]. For instance, in an FL-based recommendation system [54], [2], [39], an attacker would be interested in items liked (disliked) by a user who is a participant in FL, as the best (worst) selling items of a shopping mall is an appealing target for a commercial competitor. In this context, we ask the following questions:

Is it feasible to profile user data preference in FL? If so, can we demonstrate an effective preference profiling attack?

Preference Profiling Attack: We provide affirmative answers and demonstrate that effectively profiling user preference can be achieved by leveraging our presented techniques. A viable PPA attack is built upon our key observation.

Key Observation: A model memorizes the data-distribution characteristic of a training dataset, and inadvertently reflects it in the form of gradient changes, that is, when the model is being trained upon a dataset, its gradient change is related to the sample size of a class. For example, if data does not exist or has a small amount in the dataset, the model does not have the ability of generalization at the beginning. Thus, the model will exhibit *a greater gradient effect to change the weights of the corresponding neurons to minimize the expected loss of the model*. That is, the gradient change or sensitivity in the model training process is inversely proportional to the sample size of a class: a large (small) gradient change or sensitivity will be introduced when the sample size of a class is small (large) (detailed in Section V-A). Additionally, FL has two inherent characteristics: system and statistical heterogeneity [26], [21]. The statistical heterogeneity means that FL users have different data distribution in reality, and amplifies the gradient sensitivity discrepancies among diverse classes of the local dataset, which facilitates PPA.

Challenges: To achieve PPA, there are three main challenges:

- How to extract and quantify the gradient sensitivity of a local model per class (label)?
- How to improve the precision of the gradient sensitivity in a fine-grained manner?
- How to profile the sample size proportion of a class label given the quantified sensitivity?

Our Solutions: For the first challenge, given a user up-loaded model, we iteratively retrain it with a few samples per class to extract the gradient sensitivity (variations) per class. A larger sample size proportion of class leads to a lower sensitivity. For the second challenge, we strategically select a subset of users to aggregate a subset of models of interested users for the global model update—this is alike the sampling commonly used in FL. We call it selective aggregation. Because trivially aggregating all models will conceal per user's data characteristics, rendering difficulty of saliently extracting the local model sensitivity in the consecutive rounds. This allows the attacker to gain fine-grained sensitivity information of interested users in the next FL round while not degrading

the global model utility, thus still remaining stealthy. As for the last challenge, we leverage an attack model that is a meta-classifier to automatically predict targeted user data preference by feeding the extracted sensitivity information of user uploaded model and aggregated model as inputs.

Contributions: We have made three main contributions:

- We reveal a new type of privacy inference attack, Preference Profiling Attack (PPA), on FL. By exploiting the fact that the model memorizes the data preference (i.e., Majority (Minority) class) and leaves traces, we construct a model sensitivity extraction algorithm to determine the gradient change per class. Through the designed meta-classifier, PPA can infer the preference class of the user's local data in FL.
- We design a selective aggregation mechanism to greatly improve the success rate of PPA and alleviate the cancellation effect of intuitive global model aggregation on inference attacks. It aggregates a victim user's model with x models with the highest (lowest) model sensitivity of majority (minority) class, that is, the model sensitivity of x local models to the victim local model's majority (minority) class is opposed to that of the targeted local model. The operation can amplify the *differential model sensitivity*, significantly improving efficacy of the meta-classifier for better attack accuracy.
- We perform a comprehensive evaluation of PPA on privacy-concerned tasks, in particular, online shopping (i.e., Products-10K) and selfies sharing (i.e., RAF-DB) in addition to the MNIST and CIFAR10 based extensive validations. Experimental results affirm that PPA can accurately infer the top-k (e.g., $k = 1, 2, 3$) preferences of user datasets in FL contingent on a local user's characteristics. Our experiments demonstrate that PPA retains high efficiency when we increase the number of users in FL and the number of local training epochs adopted by the user. We evaluate PPA against the non-cryptographic defenses of dropout [43] and differential privacy [12], resulting in the conclusion that PPA is still highly effective if the global model does not allow a notable accuracy drop.

II. BACKGROUND AND RELATED WORK

In this section, we first introduce FL as well as its commonly used aggregation mechanism `FedAvg`, and then briefly describe existing privacy inference attacks.

A. Federated Learning

FL obviates the problematic raw data sharing from distributed users. In FL, each user trains local model and only uploads the model rather than any raw data to the server for computing global model of each new round. `FedAvg` [31] is the most well-known aggregation mechanism adopted by FL. In `FedAvg`, the global model θ_{agg} in round t is computed by:

$$\theta_{\text{agg}}^t = \sum_{n=1}^{N_{\text{user}}} \frac{D_n}{D} \theta_n^t, \tag{1}$$

where N_{user} users participate in FL, each holding D_n data points to train local model θ_n^t in round t. In addition, `FedAvg` usually utilizes a parameter C: the fraction of users that participate in aggregation at a given round. This is useful considering the 'straggler' due to network stability or simply

leaving of some users. When the sampling rate C is set to 0.1-0.2, FL reaches a good trade-off between computational efficiency and convergence rate, as demonstrated by [31]. Therefore, for the selective aggregation experiments in our work, we align previous work [31] by normally setting the proportion of participated users at each aggregation (i.e. $x = 1$–4 if $N_{user} = 10$) in order to obtain the best experimental effect.

B. Membership Inference Attack

The membership inference attack [41] proposed by Shokri *et al.* constructs shadow models by imitating the behavior of target model, and then trains the attack model according to their outputs, which can infer the existence of a specific data record in the training set. Salem *et al.* [40] optimized the attack by decreasing the number of shadow models from n to 1. Nasr *et al.* [34] designed a white-box membership inference attack against centralized and FL by exploiting the vulnerability of stochastic gradient descent algorithm. Zari *et al.* [57] also demonstrated the passive membership inference attack in FL. Chen *et al.* [8] provided a generic membership inference attack to attack the deep generative models and judged whether the image belongs to the victim's training set by devising a calibration technique. Leino *et al.* [24] utilized the model overfitting impact to design a white-box membership inference attack, and demonstrated that this attack outperforms prior black-box methods. Pyrgelis *et al.* [38] focused on the feasibility of membership inference attacks on aggregate location time-series, and used adversarial tasks based on game theory to infer membership information on location information. Some membership inference attacks [20], [17], [18] attacked generative model under the white-box and black-box settings.

C. Property Inference Attack

Ganju *et al.* [13] noted that fully connected neural networks are invariant under permutation of nodes in each layer and developed a property inference attack that can extract property information from the model. Melis *et al.* [32] devised a feature leakage attack in collaborative learning, which can infer properties that are independent of the joint model's target properties through holding a subset of training data. Recently, Mathias *et al.* [36] studied the impact of model complexity on property inference attacks in convolutional neural networks and the results demonstrated that the privacy leakage risk exists independently of the complexity of the target model. Gopinath *et al.* [14] proposed a property inference attack that automatically infers formal properties of feed-forward neural networks. They used encoding convex predicates on the input space to extract the input properties.

D. GAN Attack

Hitaj *et al.* [19] first proposed GAN attack against FL. The attacker disguises as a normal user to join the model training and obtains the imitation data of other users based on GAN to reconstruct prototypical images of the targeted training set. In addition, Wang *et al.* [50] proposed an inference attack based on GAN and multitask discriminator in FL, which achieves user privacy disclosure and is able to recover the private data of the target user on the server-side without interfering with the training process.
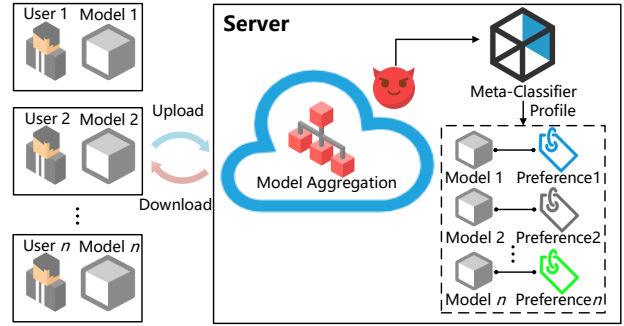

Fig. 2: Threat model of PPA.

E. Privacy Inference Attacks on FL

In FL, there have been great efforts to explore its privacy leakage through various inference attacks, including membership inference [34], [57], property inference [32] and GAN [19], [50]. These inference attacks can obtain a variety of user privacy information from the local model. However, none of these attacks can infer the *preference classes* of user dataset. Therefore, our PPA is positioned as a new type of privacy inference attack orthogonal to known ones, as summarized in Figure 1.

III. PREFERENCE PROFILING ATTACK

In this section, we first define PPA's threat model and clarify its attack goals, then present an overview of PPA, followed by PPA implementation details of each component. To ease the following description and understanding, we mainly use majority class preference profiling for descriptions. But the attack is equally applicable to the minority class preference and further extendable to top-k classes preference profiling.

A. Threat Model and Attack Goals

*1) **Threat Model**:* There are two types of entities in FL: local users and the server serving as FL coordinator. Notably, the user here is unnecessarily a human user, it could be a party or an institution e.g., a shopping mall or a hospital. Generally, the server is assumed to be malicious and launches PPA to profile a targeted user's preference through observing the victim's uploaded model, shown in Figure 2. PPA profiles the user preference class in the plaintext domain—server aggregates the local models in plaintext—rather than in the ciphertext domain, which is aligned with existing privacy inference attack counterparts [41], [34], [13], [32], [19] as summarized in Figure 1. The capabilities and knowledge of both entities are elaborated as follows:

- **Victims (Users):** Victims are the users whose private local data preferences are interested by the server or attacker. Local users participate in FL and collaboratively contribute to learning a global model without directly sharing their locally resided data. Due to statistical heterogeneity, they have different data volumes per class, thus forming diverse data distributions. We assume that users neither trust other users nor share local data and model parameters with each other. Users genuinely follow the FL's procedure

and they may employ some common privacy-enhancement techniques, particularly differential privacy and dropout.

- **Attacker (Server):** The server intends to profile a victim's preference, especially, majority information from her uploaded model, e.g., shops train the federated model based on users' shopping records, thus using PPA to analyze users' shopping habits. FL enforces privacy protection by local training instead of uploading sensitive data to the server. Therefore, in most inference attacks on FL [34], [32], [19], the server usually acts as an attacker trying to infer privacy information from the local model. We assume that the server only knows types of users' training datasets and can obtain a small set of benign samples covering all categories, e.g., from public sources, as an auxiliary dataset. The auxiliary dataset does not intersect with any user's training dataset. The assumption of accessible small public dataset in FL is aligned with [34], [61], [7], [11]. We assume that the server purposely chooses (samples) a specific subset of local models to update the global model through *model aggregation*, which is a common FL technique to mitigate adverse effects from 'straggler' users due to, e.g., unstable network or simply leaving [26], [27].

2) Attack Goals: Overall, PPA is a white-box attack, and the attacker can profile preferences of (all) participated users simultaneously to achieve the following goals:

- **Attack Efficacy.** PPA is to achieve a high attack success rate simultaneously for (all) users in FL, even if the user applies common privacy-preserving mechanisms (i.e., differential privacy and dropout).
- **Attack Stealthiness.** This is to ensure that the global model availability and utility are not degraded and its accuracy is on par with that trained via a normal FL without PPA. Since the selective aggregation mechanism we designed does not forge models and disguised under the common sampling strategy, it is difficult for users to be aware that their local models are under attack as the model utility is not affected.
- **Attack Generalization.** This goal is to ensure that our attack is generic to diverse FL settings (e.g., heterogeneous data), and varying preference profiling interests (e.g., most liked, most disliked, top-k with k no less than 1), number of FL users, and number of local training epochs chosen by individual users.

B. PPA Overview

PPA is based on the key observation that sample proportion size of classes in the training dataset can have a direct impact on a model's gradient change sensitivity. The server as an attacker exploits user model sensitivity extraction to profile user (data) preferred class(es), especially majority and minority class that to a large extent is most interested in commercial applications as we later evaluated. At the same time, the global model accessed by the user is provided without utility degradation while facilitating PPA by selectively aggregating a subset of user models, namely selective aggregation. Overall, PPA has four steps as illustrated in Figure 3, each of which is succinctly described below and then elaborated. The notations are summarized in Table I.

1) **User Local Model Training.** Users with heterogeneous data train models locally and upload them to the server.

TABLE I: Notation summary.

Notation	Description
N_{user}	Number of users in FL.
mc_n	Majority (minority) class in n_{th} user's local data.
T	Training rounds of FL.
θ_n^t	Local training model of user$_n$ in round t.
D_{aux}	Auxiliary dataset held by the server.
α	Learning rate when server extracts sensitivity.
m	Meta-classifier.
$\text{Agg}()$	Model aggregation operations.
x	Number of models selected for aggregation.
N_{label}	Number of classes in the dataset.
S_n^t	Model sensitivity of user$_n$ at round t.
DS_n^t	Differential model sensitivity of user$_n$ at round t.
Th_{round}	Discrimination round threshold.

2) **Model Sensitivity Extraction.** After receiving a user local model, the server uses the auxiliary dataset to retrain the model in order to extract the model sensitivity per class.
3) **User Preference Profiling.** When getting model sensitivity, the meta-classifier is used to predict a user preferred class(es).
4) **Selective Aggregation.** Instead of forming the global model through all user models, for a targeted local model, only x models with the highest (lowest) model sensitivity of its majority (minority) class compared to the targeted model are used in the aggregation. And then this aggregated model is sent to the targeted user—each user may receive a different aggregated model. This is to improve the PPA's attack accuracy.

Step 3) requires a meta-classifier, which is prepared during the offline phase. In other words, before FL begins among multiple users, the server trains the meta classifier offline based on auxiliary dataset D_{aux}. The implementation details of each step and the offline meta-classifier training are elaborated in the following. According to the time order, we start by meta-classifier offline training.

C. Meta-Classifier Offline Training

Algorithm 1: Training Meta-Classifier in Centralized Learning

---

1 **Input: Auxiliary dataset D_{aux}, learning rate α**
2 **Output: Meta-classifier m**
3 Sample the data in D_{aux} into multiple data distributions and train shadow models correspondingly
4 Record i_{th} shadow model parameters θ_{shadow_i} and its preference class mc_i
5 Divide D_{aux} into N_{label} retraining datasets with all classes:
$D_{\text{aux}} \rightarrow \left\{ D_{\text{aux}_1}, ..., D_{\text{aux}_t}, ..., D_{\text{aux}_{N_{\text{label}}}} \right\}$ with D_{aux_t} having samples solely from the t_{th} class.
6 Record retrained model parameters $\theta_{\text{retrain}_i}$ and compute model sensitivity S of its class
$S = \sum \left| (\theta_{\text{retrain}_i} - \theta_{\text{shadow}_i}) \cdot \frac{1}{\alpha} \right|$
7 Label the meta-data (S, mc_i), and train meta-classifier m

Before FL begins, the server needs to train a meta-classifier with the auxiliary dataset D_{aux}, as detailed in Algorithm 1. Firstly, the server samples the data in D_{aux} into a subset
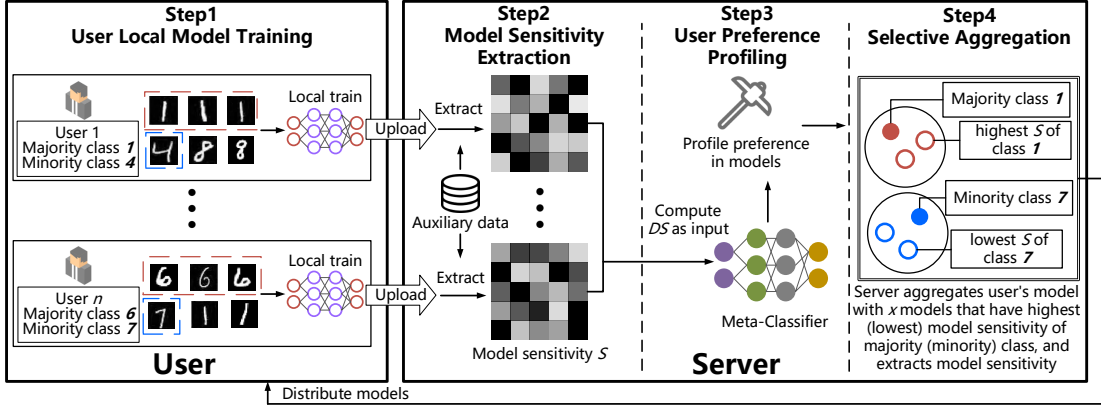
Fig. 3: An overview of PPA.

Algorithm 2: Training Meta-Classifier in Federated Learning

1 **Input: Auxiliary dataset** D_{aux}**, learning rate** α
2 **Output: Meta-classifier** m
3 Combine the data in D_{aux} into multiple data distributions and train shadow models respectively
4 Record preference class mc of each shadow model
5 Divide D_{aux} into N_{label} retraining datasets with all classes:
  $D_{\mathrm{aux}} \rightarrow \left\{ D_{\mathrm{aux}_1}, ..., D_{\mathrm{aux}_t}, ..., D_{\mathrm{aux}_{N_{\mathrm{label}}}} \right\}$ with D_{aux_t} having samples solely from the t_{th} class
6 **Shadow Model Pairing:**
7 Aggregate each shadow model with its paired one that has opposite preference class, and record the aggregated model parameters for such shadow model θ_{agg_i}
8 Retrain θ_{agg_i} on per class subset D_{aux_t} to record model parameters $\theta_{\mathrm{agg}_i}^{\mathrm{retrain}}$
9 Compute model sensitivity of this aggregated model
  $S1 = \sum \left| \left(\theta_{\mathrm{agg}_i}^{\mathrm{retrain}} - \theta_{\mathrm{agg}_i} \right) \cdot \frac{1}{\alpha} \right|$
10 **Shadow Model Updating:**
11 Update aggregated model θ_{agg_i} on dataset corresponding to the i_{th} shadow model to resemble the local model training for the next round in FL, and record updated model parameters θ_{aux_i}
12 Retrain $\theta_{\mathrm{agg}_i}^{\mathrm{retrain}}$ on per class subset θ_{aux_t} to record model parameters $\theta_{\mathrm{aux}_i}^{\mathrm{retrain}}$
13 Compute updated model sensitivity of its preference class
  $S2 = \sum \left| \left(\theta_{\mathrm{aux}_i}^{\mathrm{retrain}} - \theta_{\mathrm{aux}_i} \right) \cdot \frac{1}{\alpha} \right|$
14 Label the meta-data ($|S1 - S2|$, mc), and train meta-classifier m

---

to resemble a heterogeneous data distribution of a user. The sampling repeats a number of times. For each subset, a shadow model is trained and its preferred class is acted as the label. Secondly, the server separates D_{aux} into N_{label} subsets $D_{\mathrm{aux}} \rightarrow \left\{ D_{\mathrm{aux}_1}, ..., D_{\mathrm{aux}_t}, ..., D_{\mathrm{aux}_{N_{\mathrm{label}}}} \right\}$ where N_{label} is the total number of classes, each subset consisting of samples from a single class. Then, each shadow model is retrained per subset (D_{aux_t}) to extract shadow model sensitivity per class. Thirdly, such sensitivity (in particular, the gradient change) and its preferred class given a shadow model are used as the training set to train the meta-classifier. The meta-classifier can profile the preference class of the dataset.

However, the meta-classifier trained in the above straight-forward manner is only suitable for centralized deep learning not the collaborative learning, in particular, in FL. Because when a local model is aggregated, the data distribution characteristics will be concealed within the global model, resulting in the model sensitivity fading of each user preference class in the following round. To overcome this challenge, we improve Algorithm 1 to make the meta-classifier applicable to FL, as detailed in Algorithm 2 (the performance comparison between the baseline Algorithm 1 and Algorithm 2 are detailed in Section IV-B7). Generally, we leverage two specific improvements: paired aggregation, and differential sensitivity in two consecutive rounds.

In the meta-classifier training of FL, each shadow model will be paired with another shadow model. The steps are as below.

1) For the i_{th} shadow model with majority class mc_i, the server selects the other shadow model with the highest model sensitivity of mc_i (i.e., mc_i is the minority class to a large extent) from all rest shadow models, and aggregates it with the i_{th} shadow model.
2) The server extracts the model sensitivity $S1$ of the aggregated model, and updates each aggregated model on the dataset corresponding to each shadow model.
3) Then the model sensitivity $S2$ of the updated i_{th} shadow model in the next round is extracted.
4) The difference between $S1$ and $S2$ resembling two consecutive rounds in FL, $|S1 - S2|$, and its preferred class are meta-data used to train the meta-classifier in FL.

Note in step 1), in the case of minority, the model with the lowest model sensitivity of mc_i model is selected for pairing. Also, the $S1$ is extracted from the aggregated model, while the $S2$ is merely extracted from the updated local model in the following round.

The meta-classifier will be used to accurately profile the user's preference during the online FL learning procedure when the sensitivity of user uploaded model is extracted and fed. The input of meta-classifier is the model sensitivity difference of two consecutive rounds ($|S1 - S2|$). In order to maximize the differential model sensitivity and make the meta-classifier get more accurate classification, we propose and

leverage selective aggregation during the FL's training, which will significantly increase S1. Since S2 remains relatively unchanged regardless of whether the selective aggregation is employed, a higher differential model sensitivity will be generated in the next round to obtain accurate profiling.

D. User Local Model Training

FL is with a number of users in practice, and the users may be located in different regions, indicating statistical and system heterogeneity. The PPA-concerned statistical heterogeneity can be a result from personalized data that are collected [58]. Thus, each user's training set has its own preferred category (class). As shown in step (1) in Figure 3, each user local data thus has its unique data distribution. Each user has a majority (minority) class mc_n, that is, the largest (lowest) number of samples in the dataset. For example, label 1 and label 6 are the majority classes of user_1 and user_n, and label 4 and label 7 are the minority classes of user_1 and user_n, respectively, as exemplified in Figure 3. In FL, the local model may be trained multiple epochs before it is uploaded to the server in an FL round. PPA has naturally tolerated such variation, which is invariant to the epoch number adopted by the user (see Section IV-E2). After local training, each user updates the model parameter θ_n^t and uploads it to the server at the t_{th} round. The local model inadvertently memorizes the preference of the corresponding user, which is the private information that the meta-classifier is interested in profiling later.

E. Model Sensitivity Extraction

To quantify the model sensitivity to preference class, we utilize sum of absolute values of the gradient changes before and after the local user model retraining, which is expressed:

$$S = \sum \left| \frac{\delta}{\delta \theta_n^{t'}} L(\theta) \right| = \sum \left| \left(\theta_n^t - \theta_n^{t'} \right) \cdot \frac{1}{\alpha} \right|, \qquad (2)$$

where θ_n^t and $\theta_n^{t'}$ respectively represents the uploaded and retrained model parameter vector of user_n at the t_{th} FL round, and $L(\cdot)$ represents the loss function. Given a local model uploaded, the server retrains it based on the retrain dataset $\left\{ D_{\mathrm{aux}_1}, ..., D_{\mathrm{aux}_t}, ..., D_{\mathrm{aux}_{\mathrm{N_{label}}}} \right\}$ with D_{aux_t} to extract the sensitivity of the model per class. The θ_i' denotes the retrained model. The process is detailed in Algorithm 3.

F. Preference Class Profiling

We launch PPA in the FL's training phase. In this phase, the server leverages a trained meta-classifier to profile user preference after extracting its uploaded model sensitivity. The meta-classifier profiles preference classes based on the differential model sensitivity between consecutive rounds. There are two considerations we take to improve the attack accuracy.

Firstly, it may be inaccurate to determine the user's preference class merely according to sensitivity extracted from one FL round. Therefore, we determine the preference based on a number of consecutive rounds only if their preferences predicted by the meta-classifier are all same. This number of rounds is denoted as $\mathrm{Th}_{\mathrm{round}}$. Given meta-classifier outputs the same preference of a user in consecutively $\mathrm{Th}_{\mathrm{round}}$ FL rounds, this preference is locked, and PPA against this user

---

Algorithm 3: Extracting Model Sensitivity

1 Input: Local models of all users in the round t θ_n^t, Auxiliary dataset D_{aux}, learning rate α

2 Output: Model sensitivity of class c S_n^t

3 Set $S_n^t = 0$

4 Generate retraining set of class c: D_{aux_c}

5 foreach $\underline{\theta_n^t}$ from 1 to N_{user} in the round t **do**

6   Retrain model $\theta_n^{t'} = \theta_n^t.train(D_{\mathrm{aux}_c}, \alpha)$

7   **foreach** $\underline{\text{neuron } i \text{ in } \theta_n^t}$ **do**

8   Compute $S_{(n,i)}^t = \left| \left(\theta_n^t - \theta_n^{t'} \right) \cdot \frac{1}{\alpha} \right| = \left| \frac{\delta}{\delta \theta_i} L(\theta) \right|$

9        $S_n^t += S_{(n,i)}^t$

10   **end**

11 end

---

is completed, which means the monitoring on this user is no longer needed. The process is detailed in Algorithm 4.

Secondly, the FL's aggregation can make inference attacks [34], [32] challenging, as the global model merges all users' data characteristics. With one and more FL rounds, data characteristics of a given user are concealed and faded, thus preventing PPA from accurately extracting the user's model sensitivity that reflects the user's local data characteristics. To address this problem, we propose selective aggregation elaborated as follows.

Algorithm 4: Profiling Preference Class

1 Input: Model sensitivity S_n^t of user_n in the round t, Model sensitivity S^{t-1} of aggregated model in the last round t-1, Round threshold $\mathrm{Th}_{\mathrm{round}}$

2 Output: Preference class of user_n mc_n

3 Users train locally and upload models θ_n^t

4 Server retrains θ_n^t based on retraining dataset to get $\theta_n^{t'}$

5 Extract model sensitivity S_n^t of θ_n^t

6 Compute the differential model sensitivity between S_n^t and S^{t-1} resembling two consecutive rounds $\left| S^{t-1} - S_n^t \right|$

7 Input differential model sensitivity into meta-classifier, and profile preference class mc_n

8 if $\underline{\text{same result in } \mathrm{Th}_{\mathrm{round}} \text{ rounds}}$ **then**

9   PPA against this user is completed

10 end

11 Each uploaded model of user is aggregated with x models that have the highest (lowest) model sensitivity of its majority (minority) class

12 Extract model sensitivity S^t of each aggregated model and then distribute such aggregated model back to corresponding user

---

G. Selective Aggregation

There are two challenges of selective aggregation confronted by PPA: 1) how to prevent the user model sensitivity from being decreased or concealed by the global aggregation process after a number of FL rounds? 2) how to further amplify the user model sensitivity to reflect the user's specific local data characteristics?

In the selective aggregation, given a targeted user, the server selects models of x users who exhibit an opposite majority (minority) class as their models have the opposite highest (lowest) extracted model sensitivity—note lowest sensitivity
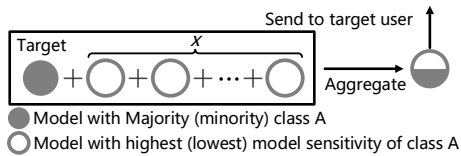
Fig. 4: Selective aggregation. The selected x users have *opposite model sensitivity* of class A that is the majority (minority) class of the targeted user.

corresponds to majority class. Then these $x + 1$ models are aggregated to form a global model, which is then sent back to the targeted user only, as shown in Figure 4—for other users who are not interested by the attacker, the global model sent to them follows a normal aggregation. After selective aggregation, the server extracts the model sensitivity of each aggregated model for the next round of meta-classifier profiling. The selective aggregation is inconspicuous to the local users due to two main facts. Firstly, users only communicate with the server rather than other users, and the communication between a user and the server is usually through a standard secure communication channel. This is important to prevent privacy leakage (e.g., property inference attack [32]) due to a victim user's local model direct exposure to a malicious user—attack occurred between users. Secondly, FL can indeed distribute different global models to users, such as personalized FL [29] that leverages a clustering technique to group users and sends different global models to different groups of users. Therefore, it is tractable to make the users unaware of selective aggregation. Later we experimentally show that the local data accuracy predicted by the global model (i.e., model utility) is similar from the user's perspective with and without implementing the selective aggregation.

The input of meta-classifier is the model sensitivity difference of two consecutive rounds. We define it as differential model sensitivity (DS), and compute as follows:

$$DS_n^t = \left| S_{\mathrm{agg}_n}^{t-1} - S_n^t \right|, \tag{3}$$

where $S_{\mathrm{agg}_n}^{t-1}$ is the model sensitivity of the *aggregated model* in round $t - 1$, and S_n^t is the model sensitivity of the updated user model in round t. Through selective aggregation, $S_{\mathrm{agg}_n}^{t-1}$ of each model will be amplified, while the S_n^t is stable compared to the case without the usage of selective aggregation. Therefore, the DS in each round is greatly improved since the meta-classifier is trained based on DS, the larger the DS, higher the profiling accuracy. Notably, in order to maximize the DS in the next round of PPA, the server utilizes selective aggregation by sending different models to each user.

IV. EXPERIMENTS

In this part, we experimentally evaluate PPA and analyze its attack performance under four data heterogeneity metrics (*CP*, *CD*, *UD*, and *ID*) in FL. In addition, we validate the effectiveness of selective aggregation under a number of varying settings. Moreover, the scalability of PPA is examined and validated.

A. Experimental Setup

In our experiments, common personal computers act as users in FL with an identical configuration (as system heterogeneity is not our concern): Intel Core i5 processors for a total of four cores running at 3.20 GHz and 12 GB RAM. We use H3C UIS 3010 G3 8SFF server as the FL's aggregator (namely the attacker), which is equipped with two Intel Xeon CPU Silver 4214, 128 GB RAM and two 480 GB SSD.

To evaluate the efficacy of PPA inference, we conduct experiments on four different datasets. In particular, the Products-10K is used to represent a commercial scenario and the facial expression dataset (i.e., RAF-DB) represents personal private information.

1) Datasets and Model Architecture: We select four datasets to evaluate PPA. MNIST and CIFAR10 are used to comprehensively show the PPA's performance under four data heterogeneity metrics and the effectiveness of selective aggregation. Products-10K and RAF-DB are real-world datasets used to indicate the PPA's performance under commercial scenarios of shopping and social network and the scalability of PPA.

- *MNIST* is an image dataset of handwritten numbers from 0 to 9 [23]. The images in the database are all 28×28 grayscale images, including 60,000 training images and 10,000 test images.
- *CIFAR10* is an image dataset for the recognition of universal objects [22]. There are 10 categories of RGB color images with size of is $32 \times 32 \times 3$. It consists of 50,000 training and 10,000 testing samples, respectively.
- *Products-10K* is a product identification set built by JD AI research, which contains about 10000 products often purchased by Chinese consumers and covers many categories, e.g., fashion, food, health care, household products, etc [4]. We use this dataset to resemble a real-world FL shopping scenario and evaluate the impact of our attack.
- *RAF-DB*, Real-world Affective Faces Database [25], is a large-scale facial expression database with around 30K great-diverse real-world facial images. Images in this dataset are of the great variability in subjects' age, gender and ethnicity, head poses, lighting conditions, occlusions, post-processing operations, etc.

The model architecture for training MNIST is a convolutional neural network with two convolution layers, followed by a max pooling layer, and then a fully connected layer. As for the remaining CIFAR10, Products-10K, and RAF-DB, the model architecture has four convolution blocks: each block has one convolutional layer and one max pooling layer, followed by two fully connected layers. We note that the model architectures are well aligned with previous orthogonal privacy inference attacks. Particularly, the most complicated model architecture that previous orthogonal privacy inference attacks [13], [19], [32], [41] used is a CNN with three convolutional layers. PPA is applicable to complicated model architectures (e.g., VGG), as validated in Section V-D.

2) Data Distribution Metrics: Four metrics defined below are used to resemble the statistical heterogeneity of local user data in FL. Generally, the first two metrics are utilized to evaluate the PPA's performance under a varying data distribution

given a user, and the last two are used to validate the applicability of the selective aggregation. Note that exhaustively experimenting all combinations of four data distributions is infeasible, we thus consider typical (common) combinations of them (the range of each metric's distribution is carefully considered e.g., each *CP*, *CD*, *UD* from 10% to 100%).

- **Class Proportion (*CP*)** represents the proportion of each class in the user dataset. Closer the value to 1, more samples in this class. The *CP* is expressed as:

$$CP = \frac{\#number\ of\ samples\ given\ a\ class}{\#dataset\ size} \quad (4)$$

- **Class Dominance (*CD*)** represents the dominance of a class in the user dataset. The numerator is the difference between the majority (minority) number and the class closest to it, expressed as below:

$$CD = \frac{\#\,|Majority(Minority)\ -\ Secondary|}{\#\,dataset\ size} \quad (5)$$

- **User Dispersion (*UD*)** represents the divergence of preferred majority classes among users in FL. Numerator is the maximum difference in the number of users with the same preference class. Take an example to ease the understanding, there are three classes: A, B, C. Out of a total of 10 users there are 5 users with A as the preference class, 3 with B and 2 with C. The $UD = (5-2)/10 = 30\%$. The numerator in this case is $5 - 2 = 3$ and the denominator is 10. The closer the *UD* to 1, the more consistent the user preferences in FL, expressed as below:

$$UD = \frac{\#\,range(user\ number)}{\#\,user\ number} \quad (6)$$

- **Imbalance Degree (*ID*)** represents the degree of user data heterogeneity in FL, and measures the variance of local user *total* dataset volume (size). More specifically, different users have different number of local samples, e.g., user A having 1,000 and user B having a differing 2,000. A larger value means a greater variance of user data volume and a higher degree of data heterogeneity, computed as follows:

$$ID = \#\,s^2\,(user\ dataset\ volume) \quad (7)$$

B. Attack Performance

Here, we use CIFAR10 and MNIST datasets for a extensive evaluation of the PPA's performance under each of the aforementioned metrics. PPA is mainly evaluated against majority class. If the majority class predicted by meta-classifier is not the true majority class of the targeted user, *even this class is predicted as the second majority class*, this PPA attempt is regarded as a failure. It should be noted that in practice, inferring the second majority class is still valuable for the attacker, e.g., the most and the second most popular items of a shopping mall. This top-k profiling accuracy with k no less than 1 is evaluated in Section IV-D.

For all experiments unless otherwise specified, we set up a total number of 10 users in FL. Each local dataset has 4,000 samples. We conduct multiple evaluations under the four metrics to entirely include the general data distribution in FL [31], [32], [34]. As for the server, it has 150 auxiliary samples per class that are non-overlapped with any user. During PPA,

TABLE II: PPA attack accuracy vs model utility.

	Attack Accuracy		Model Utility	
Dataset	Train	Test	without Attack	with Attack
MNIST	0.994	0.894	0.913	0.902
CIFAR10	0.981	0.967	0.821	0.813

the server applies selective aggregation (we compare it with normal aggregation in Section IV-C).

1) Attack Accuracy under CP: We decrease the *CP* from 100% to as low as 10%. A higher *CP* means that the preferred class has a larger number of samples, thus increasing the model sensitivity gap between this majority class and the rest classes. Therefore, the attack accuracy would be improved because of easier differentiating the majority class by the meta-classifier. The results of MNIST and CIFAR10 are displayed in Figure 5 (a), which do confirm this tendency. Meanwhile, CIFAR10 performs better than MNIST because the variation in model sensitivity becomes larger as the dataset becomes richer in features. Each cases' accuracy is beyond 90% when the *CP* is higher than 40%.

2) Attack Accuracy under CD: Here we vary the *CD* from 10% to 100% and PPA's results for MNIST and CIFAR10 are detailed in Figure 5 (b). When *CD* is close to 0%—the number of majority class is close to that of second majority class, the PPA's accuracy for the majority class is low. This is because the model sensitivity extracted for majority and second majority classes are similar, making the meta-classifier to be confused to determine the truth one. Nonetheless, as *CD* increases, the attack accuracy greatly improves.

From the experimental results under *CP* and *CD*, we can see that PPA effectively profiles a user's preference within a wide range of data heterogeneity. In addition, richer characteristics of the data (i.e., CIFAR10) indicate a more effective meta-classifier for performing PPA.

3) Attack Accuracy under UD: *UD* indicates the degree of variation in the preferences of different user groups, reflecting the overall trend of preference categories in FL. It may affect the selection of user models in each round of selective aggregation, thus we evaluate the relationship between different *UD* and attack accuracy, as shown in Figure 5 (c). Whether the group preference is more consistent (*UD* close to 1) or more discrete (*UD* close to 0), PPA can accurately profile the preference class of each user in FL.

4) Attack Accuracy under ID: To affirm that selective aggregation is suitable for FL under imbalanced data, we evaluate the performance of PPA as a function of varying *ID*, as shown in Figure 5 (d). The experimental results indicate that the data amount owned by a user does not affect the effectiveness of PPA.

Through the evaluations under *UD* and *ID*, the scalability of selective aggregation leveraged by PPA is validated.

5) Local Data Size and Class Number: In previous experiments, the local data size and number of classes in the local dataset are constant to be 4,000 and 10, respectively. We here vary these two parameters (note *CP* is retained as 40% and *CD* is retained as 50%). Specifically, for user data volume, we
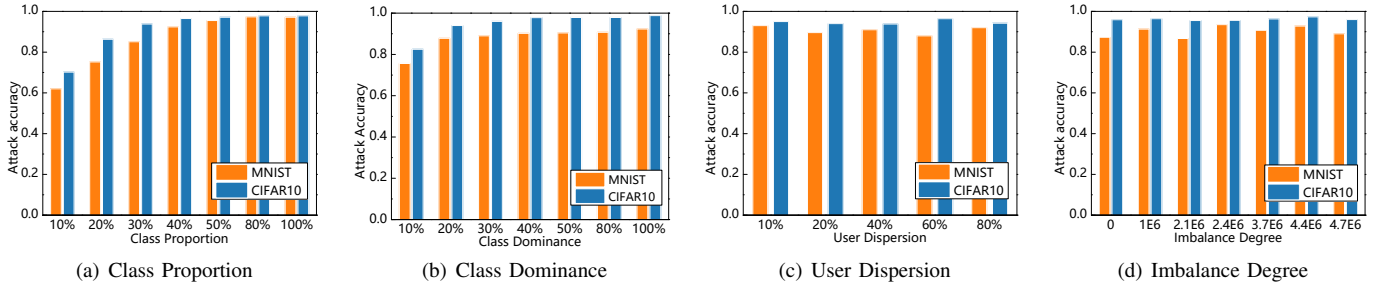
(a) Class Proportion	(b) Class Dominance	(c) User Dispersion	(d) Imbalance Degree

Fig. 5: Attack accuracy of PPA under different metrics.

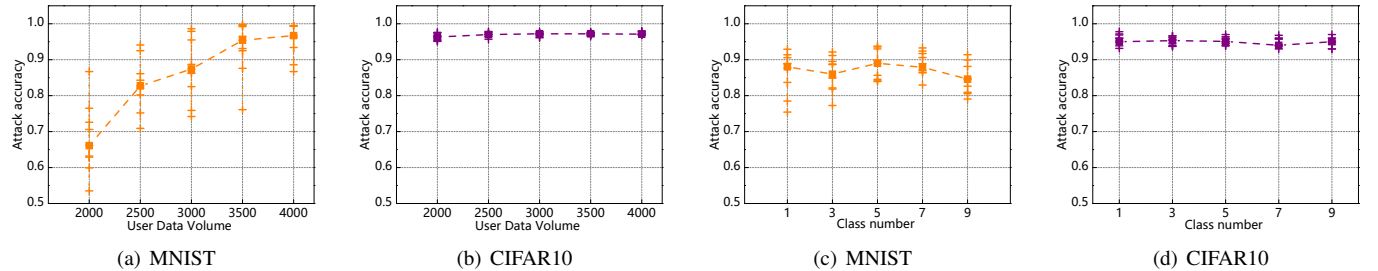|			
(a) MNIST	(b) CIFAR10	(c) MNIST	(d) CIFAR10

Fig. 6: Attack accuracy of PPA against models trained on MNIST and CIFAR10 datasets. Figure (a) and (b) reveal the attack accuracy with different amounts of data. The median values are connected across different training set sizes. Figure (c) and (d) reveal the accuracy of the attacks with different number of classes.
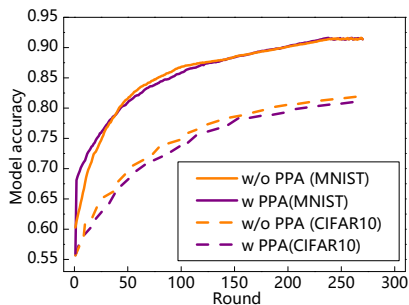
Fig. 7: A comparison of model utility w/o the attack.

vary it from 2,000 to 4,000 by a step of 500. For the number of classes owned by a user, we vary it from 1 to 9 with a step by 2. We repeat 10 times for each setting. The attack accuracy is shown in Figure 6. The results indicate that the PPA's performance is independent of the number of classes of local data. As for the local data size, it to some extent influences the PPA's accuracy, especially when the sample itself (i.e., MNIST) is with a less rich feature. However, the local data size has a very limited impact on the attack accuracy given that the CIFAR10 is with richer features.

6) Attack Accuracy vs Model Utility: To be stealthier, the global model utility downloaded by the user should not be affected by PPA. Otherwise, the user may be aware of the abnormality during the FL's training such as deviated model accuracy (utility) resulted from the selective aggregation. In this context, we evaluate the user model utility (i.e., its accuracy) in each round. In addition, we report the attack

performance.

As summarised in Table II (*CP* is 40% and *CD* is 50%), for attacking performance, we evaluate the training and testing accuracy of meta-classifier. For model utility, we compare the accuracy of the user's local model before uploading in each round with the accuracy of the received aggregation model (after attack) in the following round. The experimental results show that given the often heterogeneous data distribution, PPA has the attack accuracy of 0.894 and 0.967 in MNIST and CIFAR10. Most importantly, there is no notable model utility difference without and with attack (specifically, the applied selective aggregation), which obviates the user's awareness of PPA by solely examining the model utility. We also compare the change of user model accuracy with the number of rounds with and without the attack, as shown in Figure 7. The experimental results reveal that there is no difference in user model accuracy between the two cases because PPA does not tamper with the user model parameters. Therefore, the model with PPA does not differ notably in accuracy from normal FL that may raise suspicion among users, which satisfies attack stealthiness.

*7) Attack Performance: Algorithm 1 vs Algorithm 2:* To validate the significant performance of PPA improved from Algorithm 1 to Algorithm 2, we evaluate the loss and accuracy of meta-classifier per algorithm. The results are detailed in the Table III, which affirm that meta-classifier in Algorithm 2 exhibits greatly improved performance over Algorithm 1: lower loss and higher accuracy. Because in FL scenario, Algorithm 2 utilizes selective aggregation to extract *DS*, which is used to train meta-classifier. It is more suitable for distinguishing users with different preference classes in multiple users scenarios.
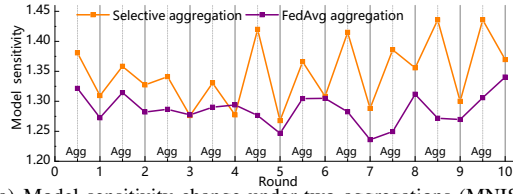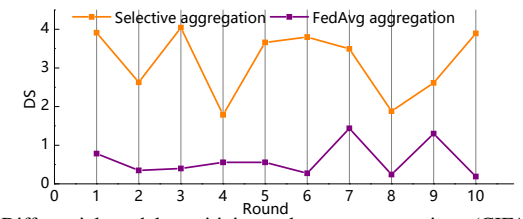
(a) Model sensitivity change under two aggregations (MNIST)
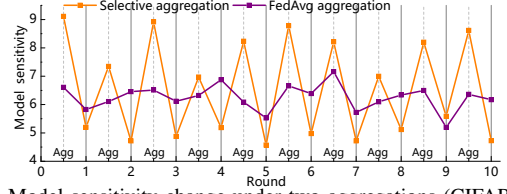
(b) Model sensitivity change under two aggregations (CIFAR10)
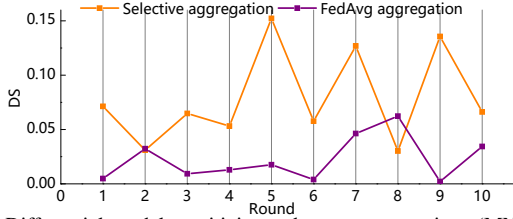
(c) Differential model sensitivity under two aggregations (MNIST)

(d) Differential model sensitivity under two aggregations (CIFAR10)

Fig. 8: Model sensitivity change and differential model sensitivity under two aggregations: with and without selective aggregation.

TABLE III: A comparison of the PPA's performance between Algorithm 1 and Algorithm 2 in terms of loss and accuracy.

	Loss		Accuracy	
	Algorithm 1	Algorithm 2	Algorithm 1	Algorithm 2
MNIST	2.23	1.34	0.43	0.85
CIFAR10	0.81	0.49	0.71	0.82
Products-10K	1.89	0.77	0.51	0.84
RAF-DB	0.88	0.64	0.66	0.81

C. w/ and w/o Selective Aggregation

So far, all experiments are with the selective aggregation. We have not validated the improved efficiency of the selective aggregation compared to the case where the selective aggregation is not enforced and instead a common aggregation algorithm (i.e., `FedAvg`) is used. In the following, we compare the PPA's performance with and without selective aggregation from four aspects.

1) Differential Model Sensitivity: Each round of FL consists of two steps: model aggregation by server and local model updating by user. In each step per round, we extract the model sensitivity of the majority class under selective aggregation and `FedAvg` aggregation respectively, as shown in Figure 8(a) and (b). Then, we also compare the differential model sensitivity under two aggregations, as shown in Figure 8(c) and (d).

As can be seen from Figure 8 (a) and (b), each round of selective aggregation increases the model sensitivity. Then the next round of local training draws down the model sensitivity again. Though the model sensitivity of the *user uploaded model*, S_{agg_n}, under `FedAvg` aggregation has no notable difference from the selective aggregation method. The model sensitivity of the aggregated model, $S_{agg_n}^{t-1}$, does have a larger difference. Since $DS_n^t = \left| S_{agg_n}^{t-1} - S_n^t \right|$, the DS_n^t of selective aggregation method is greatly amplified compared to that of the `FedAvg` aggregation method, which can be clearly observed from Figure 8(c) and (d) that the DS_n^t of selective aggregation method is always above that of the `FedAvg`.

2) Attack Performance Comparison: Figure 9 indicates the accuracy comparison of PPA under default `FedAvg` ag-
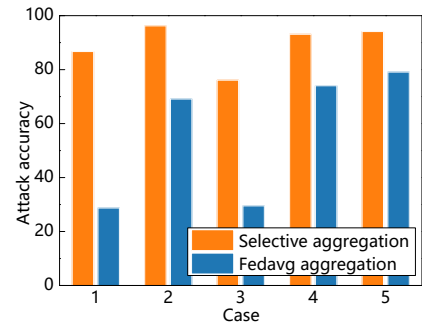

Fig. 9: The PPA's performance under common `Fedavg` aggregation and the presented selective aggregation.
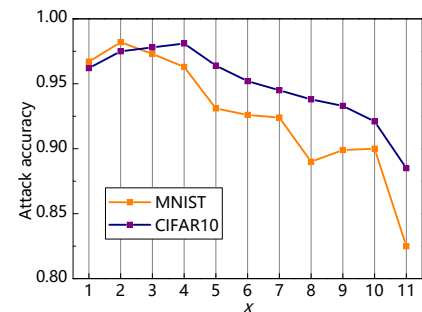

Fig. 10: Different x values in selective aggregation.

gregation [31] and selective aggregation, respectively. We set up 5 cases under these two model aggregation methods. In the experiments, there are 10 users in FL, each user has 6,000 samples of MNIST dataset, the server has 150 samples per class, *CP* and *CD* are randomly selected from 40% to 60%, and $x = 4$ in selective aggregation. The experimental results confirm that PPA has a higher accuracy under the selective aggregation compared to the default `FedAvg`, as the meta-classifier profiles the preference class according to DS. Compared with `FedAvg`, selective aggregation can improve

DS in each round and make the meta-classifier profile better.

3) Selection of Value x: Here we evaluate the PPA's performance as a function of x that is the number of models chosen in the selective aggregation (recall Figure 4). This experiment is conducted upon MNIST and CIFAR10 dataset, where the total number of users is 12 and each user has 4,000 samples. The PPA's attack accuracy with x from 1 to 11 is depicted in Figure 10. We can see that the attack is most effective when x is 1–4, and then the attack success rate decreases as the number of aggregated models increases. This is not difficult to understand, using an extreme case, when x = 11, selective aggregation is equal to the default `FedAvg` aggregation, and the attack success rate largely depends on the data distribution of specific users in FL, as shown in Figure 9. In other words, the PPA's attack accuracy for a given user under `FedAvg` aggregation is difficult to control, as the user's model sensitivity is solely dependent on its local data characteristics, where the attacker has no other means to improve its model sensitivity opposite to the selective aggregation. The intuitive aggregation of models renders less preference sensitivity change, thus making the meta-classifier more difficult to accurately profile the preference. In this context, the value of x should be properly selected according to the scale of users in FL in order to achieve optimal attacking performance.

D. Real-World Case Studies

Beyond the aforementioned MINST and CIFAR10 tasks, we leverage two more case studies to resemble real-world threats posed by PPA. In the first case, PPA attempts to profile popular sale items. In the second case, it infers the user emotions. Note here we further profile the top-k preference classes, including majority and minority classes. Because top-k preference classes can already contain sufficient privacy information about user data distribution in real scenes. Consequently, we evaluate the top-k output from the meta-classifier, and the results are summarized in Table IV. Experimental details are described below.

Note the top-k accuracy here is relatively different from the one usually used in image classification tasks. The top-k predicted classes are exactly the same as the top-k ground-truth classes, but without caring the ranking order of these k classes. For instance, the ground-truth preferred classes are A, B, C, D, E in descending order. We regard that predicted cases of $\{(A, B, C), (A, C, B), (B, A, C)\}$ in order are all acceptable and correct in our top-k setting.

1) Case I – Customer Preference Profiler: We resemble an FL scenario based on shopping records of Products-10K dataset, which contains 10 shops, each with 4,000 shopping records. The *CP* is 35% and *CD* is 32.5% in the experiment, and the server has 1,500 auxiliary data samples in total. These shops are collaboratively training a global model. In the experiment, we randomly select a shop for PPA and profile its shopping preference (e.g., bestselling items can be a private concern). We extract the top-k (i.e., $k = 2, 3$) preferences of its local dataset. The experimental results are shown in Table IV. Among the shopping records of the customer we attack, the top-3 classes are Food, Clothing and Cosmetics. The final outputs of PPA are Food, Cosmetics and Clothing, and the

TABLE IV: PPA in practical scenes.

Scene	Preference Class	PPA Output	Attack Accuracy
Shopping Scene (Products-10K)	top1-(Food)	top1-(Food)	0.78
	top2-(Food, Clothing)	top2-(Food, Cosmetics)	0.78
	top3-(Food, Clothing, Cosmetics)	top3-(Food, Cosmetics, Clothing)	0.88
Social Network (RAF-DB)	top1-(Disgusted)	top1-(Disgusted)	0.88
	top2-(Disgusted, Neutral)	top2-(Disgusted, Neutral)	1.00

TABLE V: The PPA's performance under different user numbers.

User Number	Attack Accuracy	Model Utility	
		without Attack	with Attack
10 users in FL	0.894	0.758	0.746
50 users in FL	0.820	0.754	0.753
100 users in FL	0.832	0.753	0.751

attack accuracy is 78%. It can be seen that without considering the top-3 order, PPA can infer customers' shopping habits with 78% attack accuracy. Therefore, the server can successfully infer popular items, then push relevant advertisements or give this information to its competitor.

2) Case II – User Psychological Profiler: In this experiment, the dataset of RAF-DB is used, and the *CP* is 67.5% and *CD* is 62.2%. The server intends to infer the personality hidden in the expression model published by users on social networks. For example, when PPA finds that photos of negative emotions dominate in the model uploaded by users, it can speculate that the user's personality may be negative. The experimental results are shown in Table IV. In the photo gallery of the users we attack, the two most common expressions are disgusted and neutrality. Then we can infer that the target user's personality tends to be negative. When users normally use their photos for training in FL, PPA can potentially profile the psychological character.

E. Scalability of PPA

We further investigate three crucial factors relevant to the scalability of PPA in FL. The first is the number of users or participants, where most previous experiments use value of ten. The second is the local training epochs, where all previous experiments use default one local training epoch. The third factor is the size of the public auxiliary samples reserved by the attacker to facilitate the model sensitivity extraction.

1) Number of Users: The number of users in FL we use to evaluate is 10, 50 and 100, respectively. Note that the user scale we considered, e.g., 100, is higher than existing orthogonal privacy inference attacks in FL [19], [34], [32], among which the highest user number that has been considered is 41 in [19]. We conduct this experiment upon RAF-DB dataset, in which each user has 4,000 training data, *CP* is 62.5%, *CD* is 57.5%, and 100 auxiliary sets per class. Under this setting, we attack seven users who have different preference classes. The experimental results are shown in Table V. The result validates

TABLE VI: The PPA's performance under different training epochs.

Training Epoch	PPA's Performance	Average Accuracy
Epoch Num. = 1	User1: 0.92	
	User2: 1	
	User3: 0.74	0.88
	User4: 0.86	
Epoch Num. = 20	User1: 0.88	
	User2: 1	
	User3: 0.72	0.87
	User4: 0.88	

TABLE VII: The PPA's performance under different numbers of auxiliary data.

Auxiliary Dataset	Average Accuracy
20 per class	0.675
100 per class	0.819
150 per class	0.86

that the performance of PPA is insensitive to the user number, and the attack accuracy remains to be high, 83.2%, in the case of 100 users. This is mainly because selective aggregation will aggregate the victim model with other user models that are most conducive to the attack, regardless of the number of users. In addition, we have affirmed that the model utility of global model in large-scale user scenarios is unaffected. More specifically, the model utility with PPA attack is similar to that of without PPA at various user scales (i.e., 50, 100), as shown in Table V.

2) Local Training Epochs: A local user can control the number of local training epochs being more than 1 before updating the local model to the server. Here we evaluate the PPA's sensitivity to the the number of training epochs. In the experiments, RAF-DB is used as a training task and the auxiliary dataset is 150 per class. Each user has 3000 training data, and *CP* is 62.5% and *CD* is 56%. We first set training epoch to be 1, and randomly select 4 victim users to perform PPA. Then we increase the local training epochs to be 20. The PPA's performance under 1 local training epoch and 20 local training epochs are detailed in Table VI. It can be observed from the experimental results that when users increase the number of local training epochs, the performance of PPA remains relatively stable, that is, the variance is slight without a clear tendency as the attack accuracy for user 4 increases. Thus, our empirical conclusion is that the effectiveness of PPA is independent of the epoch number of local training.

3) Auxiliary Data: The extraction of model sensitivity requires using a number of public auxiliary data. The more auxiliary data, the more comprehensive the sample features it contains, which will make the extracted model sensitivity close to the real situation. In this context, it is expected that the number of auxiliary data can affect the attack accuracy. We thus test the PPA's performance under different numbers of auxiliary data. The experimental setting is the same as that of local training epochs, and all auxiliary data in the experiment are randomly selected. We just vary the number of auxiliary data available to the server and evaluate the attack accuracy under 20, 100 and 150 auxiliary samples per class, which

experimental results are shown in Table VII. The results affirm that the more auxiliary sets, the better PPA's performance. Nonetheless, when there are few auxiliary sets (20 per class), our attack still has 67.5% accuracy (the guessing accuracy is 14.3%). According to our observation, PPA can perform well when the number of auxiliary sets is about 100 per class in our experiments.

V. DISCUSSION

We further discuss the insights of PPA with more experiments. In addition, we demonstrate the effectiveness of PPA even when common privacy protection mechanisms (in particular, differential privacy and dropout) are applied by local users. We then evaluate the overhead of PPA, followed by demonstrating its practicality when attacking complicated models.

A. *The Rationale of PPA*

Generally, we recognize that the user's model can remember the preference class, majority (minority) class, of the dataset during training, and reflects it *in the form of gradient change*. The gradient value of neural network determines the convergence direction of the model, which is an important index to make the model tend to fit. We find that it can be exploited to infer whether the model is sensitive to certain data. To validate it, we have done a series of preliminary and detailed experiments, which are elaborated in the following.

To simplify the investigation, we focus on a binary classification task. We change the data distribution of the training set of binary classifiers based on MNIST dataset, and extract the gradient change when they are later retrained by two kinds of samples A and B, respectively. Specifically, we firstly adjust the proportion of A and B in training set from 1:9 gradually to 9:1, and consequentially train 9 binary classifiers, in which the total amount of training sets of each model is 6,000. After model training, we retrain the model with A and B samples respectively, and record the model sensitivity of class A and B between the two models, as shown in Figure 11 (a). The model sensitivity is linearly related to the sample size of the given class.

Beyond the simplified binary classification, we also extract model sensitivities for each class in the MNIST and CIFAR10 models, as shown in Figure 11 (b) and Figure 11 (c). In this experiment, we train two models based on MNIST and CIFAR10 datasets, respectively. The training set is a random sample of 6,000 data to represent the data distribution of FL users. The majority class *1* and *Bird* account for 50%, minority class *8* and *Airplane* accounts for 2%, and the rest is evenly distributed. We affirm that the majority and minority classes correspond to the minimum and maximum values of the model sensitivity, respectively.

For further visualization, we extract the model sensitivity of majority and minority class in a model, by randomly selecting 36 neurons to show their gradient changes after retraining with a 6×6 square grayscale plot. The neurons are selected from the last convolutional layer of the model, as the gradients sensitivity is more salient in layers that are closer to the output than that of layers closer to the input due to the gradient decrease in the back-propagation. The larger the value of gradient
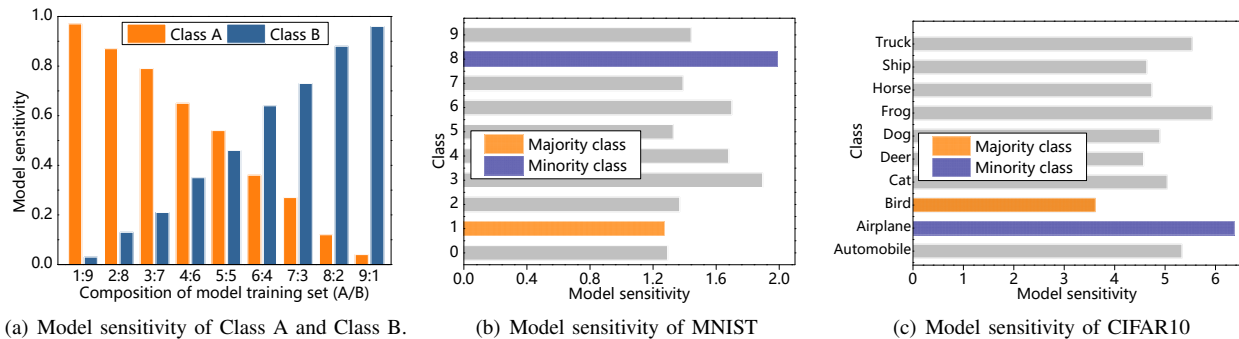
(a) Model sensitivity of Class A and Class B.　　(b) Model sensitivity of MNIST　　(c) Model sensitivity of CIFAR10

Fig. 11: Model sensitivity change and extraction experiments.
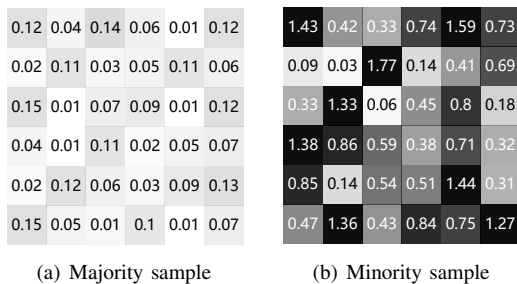

(a) Majority sample　　　　(b) Minority sample

Fig. 12: The gradient change of 36 randomly selected neurons after retraining by samples of majority (a) and minority (b).

change, the darker the color, as depicted in Figure 12. We can see that the gradient change is much smaller retrained by the majority class samples (Figure 12 (a)) than that by the minority class samples (Figure 12 (b)). The gradient value of neural network determines the convergence direction of the model, which is an important index to direct the model to fit, and can essentially reflect whether the model is sensitive to certain data distribution characteristics in our PPA. More specifically, each sample in the training set contributes to minimizing the model loss through stochastic gradient descent algorithm. For the data point(s) that does not exist or have a small amount in the training set, the model does not have a good generalization, especially, at the beginning of the model training. Therefore, this data point(s) will have a greater gradient effect to change the weights of the corresponding neurons to minimize the expected loss of the model. As a matter of fact, this is also an important motivation for taking model aggregation in FL: when users have various data distributions, the server can still make the global model converge to all data categories by aggregating all the local models. Thus, the natural data heterogeneity characteristic in FL is vulnerable to the newly devised preference inference attack.

B. Defenses

Dropout [43] and differential privacy [12], [1] are two widely used techniques in the fields of formal verification [48], program synthesis [49] and deep learning privacy protection [51], [46], [56]. Here we investigate the attack accuracy and model utility when they are adopted in FL that is under the

attack of PPA. The experimental results detailed in Table VIII demonstrate that the accuracy of PPA's meta-classifier drops when a smaller (i.e., the smaller, the stronger) privacy budget of ϵ of differential privacy is applied, and remains stable when the dropout is applied (i.e., CIFAR10). In the former case, even with a privacy budget as low as 1.21, our attacks still have a accuracy up to 57% (guessing accuracy is 10%).

However, those protection mechanisms badly affect model utility or accuracy, as we have evaluated and shown in Table VIII. It can be seen from the experimental results that the model utility (accuracy) degrades: a higher protection level (smaller ϵ) renders a lower model utility. Though the dropout and differential privacy can mitigate the user preference leakage to some extent, they are with a notable undesired utility trade-off.

In addition, the client may check whether the global models distributed to other clients are the same, since the selective aggregation used by PPA sends different global models to different users. However, this check might not always be applicable. On one hand, a normal user is unlikely to see the global models sent to other users, as a standard secure communication channel between each user and the server is enforced to prevent a user's model from being exposed to other users. Otherwise, one malicious user can leak, for example property privacy, of other users [32]. On the other hand, recent personalized FL such as [29] leverages a clustering technique to group users and indeed sends different global models to different groups of users by design.

To significantly mitigate privacy leakage in FL, cryptographic-based FL can be leveraged, where local models are aggregated in ciphertext. In this context, existing privacy inference attacks, such as [41], [34], [13], [32], [19], and our proposed PPA cannot be immediately mounted. However, using a cryptographic approach will substantially increase the computational and communication overhead, preventing users with low computing resources and limited bandwidth from participating in FL.

C. Computational Overhead

We have evaluated the computational overhead of the entire PPA attack life cycle, as detailed in the Table IX. In this experiment, ten users are participated in FL, each owns 6,000 data. The auxiliary set is 200 samples per class, and the model

TABLE VIII: PPA attack accuracy and user model utility under defense settings. We experiment with Differentially-Private Stochastic Gradient Descent (DP-SGD) [1], the most representative DP mechanism for protecting machine learning models. We set $\epsilon_1 = 96.9$, $\epsilon_2 = 19.38$, $\epsilon_3 = 4.85$, $\epsilon_4 = 1.21$ under the constraints of noise multiplier = 0.05, 0.25, 1 and 4 respectively, and $\delta = 10^{-5}$.

	Task	No Def.	Drop.	Differential Privacy			
				ϵ_1	ϵ_2	ϵ_3	ϵ_4
Attack Acc.	MNIST	0.89	0.85	0.86	0.75	0.24	0.1
	CIFAR10	0.97	0.96	0.85	0.79	0.73	0.57
Model Utility	MNIST	0.90	0.87	0.84	0.81	0.32	0.17
	CIFAR10	0.80	0.62	0.75	0.64	0.62	0.55

TABLE IX: Computational overhead of PPA (time in seconds).

Task		MNIST	CIFAR10	Products -10K	RAF -DB
Shadow model training (offline)		839.29	2220.59	4725.21	3260.96
Meta-classifier training (offline)		47.65	38.44	35.45	33.51
Model sensitivity extraction		27.19	40.24	173.74	53.86
Attack time		0.37	0.36	0.34	0.27
Selective aggregation		0.001	0.006	0.02	0.22
Local training	epoch=1	0.75	1.13	2.07	3.35
	epoch=5	1.52	2.18	5.19	5.22
	epoch=10	2.01	3.46	7.71	9.13

learning rate is 0.001. The shadow model training time and meta-classifier training time dominate the computational overhead, because shadow models have to be established for each class. More accurate shadow model requires longer training time, while improving PPA online attack accuracy (as shown in Table VI). However, we should emphasize that these time-consuming preparations are carried out by the server in the *offline* phase, without affecting the online training experience of FL when normal users participate the process. During online phase, the computational overhead of model sensitivity extraction is greater than that of other online operations. Firstly, the online model sensitivity extraction can be expedited by leveraging additional computation resources (e.g., parallelization) or future optimizations on the extraction process. Most importantly, this latency (waiting time) can be essentially disguised in real scene due to the inevitable heterogeneous computing resources and communication bandwidth available among FL users. There are always notable but normal latency caused by limited computation resource [47] and communication bandwidth restrictions (e.g., uploading (downloading) the local (global) models) [60], [28]. This normal latency will be prolonged when the number of users in FL is scaled up. Thereby, the users are expected to tolerate latency to a large extent.

D. *Complicated Model*

More complicated models can be vulnerable to PPA. Here we consider the VGG16. This experiment is based on the CIFAR10 dataset, where each user has 5,000 training data, CP

is 60%, CD is 57.5%. The server has 800 samples per class as auxiliary data. The experimental results depict that PPA can achieve 88% top-1 attack accuracy in average for all ten users. We observe that the averaged accuracy slightly drops about 8% compared with a four layer convolutional network that is used. The potential reason is that the VGG16 has about 14,714,688 parameters in total (2,359,808 parameters in last convolutional layer accounting for a fraction of 16.04%) , while the four layer model has 1,448,266 parameters (295,168 parameters in the last convolutional layer accounting for a fraction of 20.38%). When the local model sensitivity is extracted, only the last convolutional layer is utilized. This means that a smaller fraction of model parameters is exploited in VGG16, which can result in degraded sensitivity, thus rendering a slightly lower profiling accuracy of PPA. The profiling accuracy can be expected to be improved for large models when a higher proportion of parameters are leveraged to extract the sensitivity. Nonetheless, relying on the sensitivity extracted from a single layer in a complicated model, in particular VGG16, PPA still achieves a high top-1 accuracy of 88%, demonstrating its practicality of attacking large models.

VI. CONCLUSION

This work uncovered a new type of privacy inference attack (i.e., PPA) on FL to infer the preference classes of users. Based on our key observation that the gradient change reflects the sample size of a class in the dataset, we proposed techniques to quantify and extract such change as model sensitivity to profile the preference class assisted with the meta-classifier. Extensive experiments have shown that PPA is effective under various settings and poses real threats to preference-sensitive applications in real world, substantiated by two commercial applications. In addition, we have shown that PPA can be scalable in that it is insensitive to the number of FL participants and local training epochs employed by the user. While squarely using common privacy protection schemes can mitigate PPA, they introduce unacceptable model utility degradation.

VII. ACKNOWLEDGEMENTS

REFERENCES

[1] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2016, pp. 308–318. V-B, VIII

[2] V. W. Anelli, Y. Deldjoo, T. D. Noia, A. Ferrara, and F. Narducci, "How to put users in control of their data in federated top-n recommendation with learning to rank," in Proceedings of ACM/SIGAPP Symposium on Applied Computing, SAC, 2021, pp. 1359–1362. I

[3] V. Ayora, F. E. A. Horita, and C. Kamienski, "Profiling online social network platforms: Twitter vs. Instagram," in Proceedings of Hawaii International Conference on System Sciences, HICSS, 2021, pp. 1–10. I

[4] Y. Bai, Y. Chen, W. Yu, L. Wang, and W. Zhang, "Products-10K: A large-scale product recognition dataset," arXiv preprint arXiv:2008.10545, 2020. IV-A1

[5] S. Z. Béguelin, L. Wutschitz, S. Tople, V. Rühle, A. Paverd, O. Ohrimenko, B. Köpf, and M. Brockschmidt, "Analyzing information leakage of updates to natural language models," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM, 2020, pp. 363–375. I

[6] K. Buraya, A. Farseev, A. Filchenkov, and T. Chua, "Towards user personality profiling from multiple social networks," in Proceedings of AAAI Conference on Artificial Intelligence, AAAI, 2017, pp. 4909–4910. I

[7] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in Proceedings of Network and Distributed System Security Symposium, NDSS, 2021. III-A1

[8] D. Chen, N. Yu, Y. Zhang, and M. Fritz, "GAN-Leaks: A taxonomy of membership inference attacks against generative models," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2020, pp. 343–362. II-B

[9] P. Chhikara, P. Singh, R. Tekchandani, N. Kumar, and M. Guizani, "Federated learning meets human emotions: A decentralized framework for human-computer interaction for IoT applications," IEEE Internet of Things Journal,, vol. 8, no. 8, pp. 6949–6962, 2021. I

[10] China, "China data security law," https://www.china-briefing.com/news/a-close-reading-of-chinas-data-security-law-in-effect-sept-1-2021/, 2021, accessed Sep 21, 2021. I

[11] Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng, "FLOD: Oblivious defender for private byzantine-robust federated learning with dishonest-majority," in Proceedings of European Symposium on Research in Computer Security ESORICS, vol. 12972, 2021, pp. 497–518. III-A1

[12] C. Dwork, "Differential privacy," in Proceedings of Automata, Languages and Programming, 33rd International Colloquium, ICALP, vol. 4052, 2006, pp. 1–12. I, V-B

[13] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2018, pp. 619–633. I, II-C, III-A1, IV-A1, V-B

[14] D. Gopinath, H. Converse, C. S. Pasareanu, and A. Taly, "Property inference for deep neural networks," in Proceedings of IEEE/ACM International Conference on Automated Software Engineering, ASE, 2019, pp. 797–809. II-C

[15] Y. Guo, F. Liu, Z. Cai, H. Zeng, L. Chen, T. Zhou, and N. Xiao, "PRE-FER: Point-of-interest recommendation with efficiency and privacy-preservation via federated edge learning," Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, vol. 5, no. 1, pp. 13:1–13:25, 2021. I

[16] I. Hagestedt, M. Humbert, P. Berrang, I. Lehmann, R. Eils, M. Backes, and Y. Zhang, "Membership inference against DNA methylation databases," in Proceedings of IEEE European Symposium on Security and Privacy, EuroS&P, 2020, pp. 509–520. I

[17] J. Hayes, L. Melis, G. Danezis, and E. D. Cristofaro, "LOGAN: Membership inference attacks against generative models," in Proceedings on Privacy Enhancing Technologies., 2019, pp. 133–152. II-B

[18] B. Hilprecht, M. Härterich, and D. Bernau, "Monte carlo and reconstruction membership inference attacks against generative models," in Proceedings on Privacy Enhancing Technologies., 2019, pp. 232–249. II-B

[19] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2017, pp. 603–618. I, II-D, II-E, III-A1, III-A1, IV-A1, IV-E1, IV-E1, V-B

[20] H. Hu and J. Pang, "Membership inference attacks against GANs by leveraging over-representation regions," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2021, p. 2387–2389. II-B

[21] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings et al., "Advances and open problems in federated learning," arXiv preprint arXiv:1912.04977, 2019. I

[22] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2012. IV-A1

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE 1998, 2021, pp. 86(11):2278–2324. IV-A1

[24] K. Leino and M. Fredrikson, "Stolen memories: Leveraging model memorization for calibrated white-box membership inference," in Proceedings of USENIX Security Symposium, USENIX Security, 2020, pp. 1605–1622. II-B

[25] S. Li, W. Deng, and J. Du, "Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild," in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR 2017. IEEE, 2017, pp. 2584–2593. IV-A1

[26] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," IEEE Signal Processing Magazine, vol. 37, no. 3, pp. 50–60, 2020. I, III-A1

[27] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in Proceedings of International Conference on Learning Representations, ICLR, 2020. III-A1

[28] F. Malandrino and C. Chiasserini, "Toward node liability in federated learning: Computational cost and network overhead," IEEE Communications Magazine, vol. 59, no. 9, pp. 72–77, 2021. V-C

[29] Y. Mansour, M. Mohri, J. Ro, and A. T. Suresh, "Three approaches for personalization with applications to federated learning," arXiv preprint arXiv:2002.10619, 2020. III-G, V-B

[30] P. Mayer, Y. Zou, F. Schaub, and A. J. Aviv, "Now I'm a bit angry: Individuals' awareness, perception, and responses to data breaches that affected them," in Proceedings of USENIX Security Symposium, USENIX Security, 2021, pp. 393–410. I

[31] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Proceedings of International Conference on Artificial Intelligence and Statistics, AISTATS, vol. 54, 2017, pp. 1273–1282. I, II-A, II-A, II-A, IV-B, IV-C2

[32] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in Proceedings of IEEE Symposium on Security and Privacy, SP, 2019, pp. 691–706. I, II-C, II-E, III-A1, III-A1, III-F, III-G, IV-A1, IV-B, IV-E1, V-B, V-B

[33] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," Future Generation Computer Systems, vol. 115, pp. 619–640, 2021. I

[34] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in Proceedings of IEEE Symposium on Security and Privacy, SP, 2019, pp. 739–753. I, II-B, II-E, III-A1, III-A1, III-A1, III-F, IV-B, IV-E1, V-B

[35] N. Papernot, M. Abadi, Ú. Erlingsson, I. J. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," in Proceedings of International Conference on Learning Representations, ICLR 2017, 2017. I

[36] M. P. M. Parisot, B. Pejo, and D. Spagnuelo, "Property inference attacks on convolutional neural networks: Influence and implications of target model's complexity," arXiv preprint arXiv:2104.13061, 2021. II-C

[37] E. Parliament and the Council of the European Union, "General data protection regulation," https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679, 2020. I

[38] A. Pyrgelis, C. Troncoso, and E. D. Cristofaro, "Knock knock, who's there? Membership inference on aggregate location data," in Proceedings of Network and Distributed System Security Symposium, NDSS, 2018. II-B

[39] J. Qin, B. Liu, and J. Qian, "A novel privacy-preserved recommender system framework based on federated learning," in Proceedings of

International Conference on Software Engineering and Information Management, ICSIM, 2021, pp. 82–88. I

[40] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-Leaks: Model and data independent membership inference attacks and defenses on machine learning models," in Proceedings of Network and Distributed System Security Symposium, NDSS, 2019. II-B

[41] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in Proceedings of IEEE Symposium on Security and Privacy SP, 2017, pp. 3–18. I, II-B, III-A1, IV-A1, V-B

[42] A. Singh, T. Sen, S. Saha, and M. Hasanuzzaman, "Federated multi-task learning for complaint identification from social media data," in Proceedings of ACM Conference on Hypertext and Social Media, HT, 2021, pp. 201–210. I

[43] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014. I, V-B

[44] U. States, "California privacy rights act," https://www.cookiebot.com/en/cpra/, 2020, accessed Feb 05, 2021. I

[45] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in Proceedings of IEEE Conference on Computer Communications, INFOCOM, 2019, pp. 1387–1395. I

[46] M. C. Tschantz, S. Sen, and A. Datta, "SoK: Differential privacy as a causal property," in Proceedings of IEEE Symposium on Security and Privacy, SP, 2020, pp. 354–371. V-B

[47] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," arXiv preprint arXiv:1812.00564, 2018. V-C

[48] Y. Wang, Z. Ding, D. Kifer, and D. Zhang, "CheckDP: An automated and integrated approach for proving differential privacy or finding precise counterexamples," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2020, pp. 919–938. V-B

[49] Y. Wang, Z. Ding, Y. Xiao, D. Kifer, and D. Zhang, "DPGen: Automated program synthesis for differential privacy," in Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2021, pp. 393–411. V-B

[50] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in Proceedings of IEEE Conference on Computer Communications, INFOCOM, 2019, pp. 2512–2520. I, II-D, II-E

[51] N. Wu, F. Farokhi, D. B. Smith, and M. A. Kâafar, "The value of collaboration in convex machine learning with differential privacy," in Proceedings of IEEE Symposium on Security and Privacy, SP, 2020, pp. 304–317. V-B

[52] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, "Data security issues in deep learning: Attacks, countermeasures, and opportunities," IEEE Communications Magazine, vol. 57, no. 11, pp. 116–122, 2019. I

[53] C. Yang and K. Chang, "Relationship profiling over social networks: Reverse smoothness from similarity to closeness," in Proceedings of SIAM International Conference on Data Mining, SDM, 2019, pp. 342–350. I

[54] L. Yang, B. Tan, V. W. Zheng, K. Chen, and Q. Yang, "Federated recommendation systems," in Federated Learning. Springer, 2020, pp. 225–239. I

[55] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," ACM Transactions on Intelligent Systems and Technology TIST, vol. 10, no. 2, pp. 1–19, 2019. I

[56] L. Yu, L. Liu, C. Pu, M. E. Gursoy, and S. Truex, "Differentially private model publishing for deep learning," in Proceedings of IEEE Symposium on Security and Privacy, SP, 2019, pp. 332–349. V-B

[57] O. Zari, C. Xu, and G. Neglia, "Efficient passive membership inference attack in federated learning," arXiv preprint arXiv:2111.00430, 2021. II-B, II-E

[58] S. Zawad, A. Ali, P. Chen, A. Anwar, Y. Zhou, N. Baracaldo, Y. Tian, and F. Yan, "Curse or redemption? How data heterogeneity affects the robustness of federated learning," in Proceedings of AAAI Conference on Artificial Intelligence, AAAI 2021, 2021, pp. 10 807–10 814. III-D

[59] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," IEEE Communications Surveys Tutorials, vol. 21, no. 3, pp. 2224–2287, 2019. I

[60] J. Zhang, N. Li, and M. Dedeoglu, "Federated learning over wireless networks: A band-limited coordinated descent approach," in Proceedings of IEEE Conference on Computer Communications, INFOCOM, 2021, pp. 1–10. V-C

[61] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," arXiv preprint arXiv:1806.00582, 2018. III-A1

[62] C. Zhou, A. Fu, S. Yu, W. Yang, H. Wang, and Y. Zhang, "Privacy-preserving federated learning in fog computing," IEEE Internet of Things Journal, IOTJ,, vol. 7, no. 11, pp. 10 782–10 793, 2020. I

[63] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? Uncovering the data leakage in cloud from mobile apps," in Proceedings of IEEE Symposium on Security and Privacy, SP, 2019, pp. 1296–1310. I