2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00086

Achieving Consensus in Privacy-Preserving Decentralized Learning

Liyao Xiang*, Lingdong Wang*, Shufan Wang*, Baochun Li†
* John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, Shanghai, China
† Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada

***Abstract*—Machine learning algorithms have been widely deployed on decentralized systems so that users with private, local data can jointly contribute to a better generalized model. One promising approach is *Aggregation of Teacher Ensembles,* which transfers knowledge of locally trained models to a global one without releasing any private data. However, previous methods largely focus on privately aggregating the local results without concerning their validity, which easily leads to erroneous aggregation results especially when data is unbalanced across different users. Hence, we propose a *private consensus protocol* — which reveals nothing else but the label with the highest votes, in the condition that the number of votes exceeds a given threshold. The purpose is to filter out undesired aggregation results that could hurt the aggregator model performance. Our protocol also guarantees differential privacy such that any adversary with auxiliary information cannot gain any additional knowledge from the results. We show that with our protocol, we achieve the same privacy level with an improved accuracy compared to previous works.**

***Index Terms*—Differential privacy, decentralized machine learning.**

I. Introduction

Designed for challenging artificial intelligence problems, the recent proliferation of *machine learning* has powered many data analytic applications, such as image classification, object recognition, and natural language processing. To pursue a highly accurate model with better generalization capability, it is beneficial to perform training over a larger dataset or a union of datasets from heterogeneous sources. However, the datasets are often proprietary and are not allowed to be shared. For instance, to learn the health status of the general public in an area, it is required to jointly train a model over the union of private datasets belonging to biomedical companies, research institutions, or hospitals. Without any sharing, the model trained on each individual source may be inaccurate or even biased since the data is most likely homogeneous.

Many computational frameworks have been proposed for machine learning, in particular, deep learning, to allow a number of users to train local models on highly sensitive data and share the knowledge learned. A key challenge in such a framework is to prevent any private data from being leaked by the local models, as it has been known that models are capable of memorizing a lot of input information. One promising approach is *Aggregation of Teacher Ensembles* [1], [2]: each local training party, acting as a 'teacher,' trains its own model over the private dataset. The aggregator, the 'student,' sends queries to the teachers on a public dataset (usually unlabeled) and choose the label that the majority of users pick. During the entire training process, the aggregator does not have direct access to any user data.

Unfortunately, users' responses to queries pose as a great threat to their privacy. The aggregator servers, as well as other users, could be potentially adversarial and interested in peeking the user's data. Thus it is a practice for the aggregator to perform secure aggregation on responses. However, the revealing of the aggregated responses still leaks too much private information: the aggregator can be adversarial and it is legitimate to send queries to one particular user. Then the 'aggregation' would completely expose the user's response. In fact, when the aggregated sum does not meet a certain threshold, the aggregation result should be discarded — it is very likely that most users do not agree with each other since local datasets can be highly unbalanced — and thus a consensus is not achieved across the majority of users.

The requirement raises a contradiction: on one hand, the aggregation results should not be exposed to avoid leaking individual information, on the other hand, such aggregation results are needed to check if the majority of users have met a consensus. Beyond checking consensus, it is often desired that minimal information should be revealed during the computation, and hence only the final output — the label with the highest vote can be released, but not other labels which rank below. One can easily tell the scenario cannot be handled by a single crypto primitive, or a naive combination of them: while the secure sum protocol aggregates users' responses, it cannot check if the results surpass a threshold in blind; the secure comparison protocol can compare pairs of encrypted values, but cannot help revealing the comparison result.

The situation is worsened when we enforce differential privacy guarantee of the final output. If it depends on the aggregator to apply the differential privacy mechanism to the aggregated result, the additive noise would have been known, and thus the true aggregation result would be revealed. If it relies on each user to distributedly apply the differential privacy scheme, the overall noise level would be too overwhelming

*Liyao Xiang (xiangliyao08@sjtu.edu.cn) is the corresponding author with the John Hopcroft Center, Shanghai Jiao Tong University, China.

*This work was partially supported by National Natural Science Foundation of China (61902245), and the Science and Technology Innovation Program of Shanghai (19YF1424500).

resulting in significantly degraded utility performance.

Therefore, we propose a *private consensus protocol*, which is a careful choreography of several cryptographic techniques, in the context of decentralized learning. The protocol first sums up all users' values without revealing each of them, and compares the results in blind to see if the highest one exceeds the threshold. To prevent other labels rather than the highest one are revealed, we design a *Blind-and-Permute* protocol to hide the ranking and a *Restoration* protocol to reveal the ranking when necessary. To guarantee differential privacy, we adopt *Sparse Vector Technique* and *Report Noisy Maximum* to ensure minimal amount of information is leaked and thus a precise amount of noise is added. The overall result is an improved aggregation utility, while preserving individual data privacy.

Highlights of our contributions are as follows. First, we design a security protocol which allows user aggregation, threshold checking, and ranking in blind. Only if the aggregated votes exceeds a given threshold, can the label corresponding to the highest vote be revealed, otherwise nothing else is learned. Second, a differential privacy mechanism is designed to work with our private consensus protocol, such that a precise amount noise is used to achieve an improved utility. Finally, we implement our protocol based on several crypto primitives and within a machine learning framework. The experimental results on various datasets show improved accuracy performance of the aggregator model under the same privacy guarantee.

II. Related Work

The most related works fall into the following categories.

Privacy-preserving multi-party learning. For general classification tasks, Pathak *et al.* [3] aims at composing a differentially-private aggregated classifier by combing several locally trained classifiers. The resulting classifier is a perturbed average of all local classifiers. Their threat model is similar to ours in that the participating users are mutually untrusting and they depend on an untrusted aggregator to collect their local results. However, their excess risk of the perturbed aggregate model is sensitive to the increase of the participating users and thus not scalable. To overcome the drawback, Rajkumar *et al.* [4] introduce a differentially private algorithm which directly optimizes a perturbed form of the overall multi-party objective. It improves the model's generalization performance compared to [3] but at the sacrifice of privacy guarantee. Recently, Bayesian learning on distributed data is discussed in [5] where a secure multi-party sum function is used for aggregation. These three works share a common feature that the aggregator sum local data (model parameters, noisy gradients, or data summaries) in a cryptographically secure manner.

The privacy-preserving protocols for deep learning draws a lot of attention recently. In [6], Shokri *et al.* designed a distributed selective stochastic gradient descent mechanism such that each user trains on its private dataset but exchanges a fraction of the parameters with others through a global parameter server. The global model suffers significant accuracy loss as a result that each user overwhelmingly perturbs their uploaded gradients to satisfy the individual privacy requirement.

In [7], [8], homomorphic encryption algorithms are used to securely aggregate the noisy gradients from different parties in an asynchronous stochastic gradient descent update. Zhang *et al.* provide a threshold scheme [7] which ensures at least a certain number of users contribute their gradients. However, they did not check if the aggregated results surpass a threshold, nor is their scheme differentially-private. Our problem is harder as the aggregator needs to compare an unknown aggregation result with a given threshold.

Secure comparison. By the secure comparison protocol, the only output revealed is a single bit indicating if the value possessed by one party is larger than the other but nothing else. There is a long line of work showing how two or more parties can securely compare their (shared) secret data, such as bitwise less-than protocols [9]–[11], protocols based on homomorphic encryption [12], [13], a non-interactive protocol based on Goldwasser-Micali encryption scheme [14], a hybrid approach of additively homomorphic system and garbled circuit [15], etc. We adopt the homomorphic encryption in [12], [13] for that it minimizes the amount of data users send.

Privacy via distributed noise generation. Works on the distributed noise generation to provide differential privacy guarantee are also related to this work. If a trusted aggregator exists, it can add calibrated noise to the aggregation result to ensure privacy. Problems arise as when such a trusted aggregator does not exist, differential privacy has to be guaranteed through a distributed protocol. Two categories of solutions are representative: interactive protocols and non-interactive ones with an untrusted server. In the former such as [16], [17], Dwork *et al.* develop a general distributed algorithm for Gaussian noise generation with fewer executions of verifiable secret sharing. Although their result is provably better than non-interactive solutions in terms of accuracy, the protocol is not very practical as all users need to be simultaneously online and interact periodically. In the latter ones, users alter their data via *randomized response* before sending it to the server (aggregator in this paper). Our differential privacy mechanism falls into the non-interactive category.

III. Preliminaries

For ease of understanding, we first introduce some background knowledge of this work.

A. Semi-Supervised Knowledge Transfer

Our work is based on a decentralized machine learning framework called semi-supervised knowledge transfer. An illustration of the framework is shown in Fig. 1. The framework consists of a number of users and an aggregator who would like to train a joint model over all users' data. Each user owns a private dataset and runs learning algorithms locally. The aggregator, who cannot access to users' data, only has an incomplete public dataset which is usually unlabeled. When local training is done, the aggregator sends queries about the unlabeled instance to each user. Then users transfer the knowledge

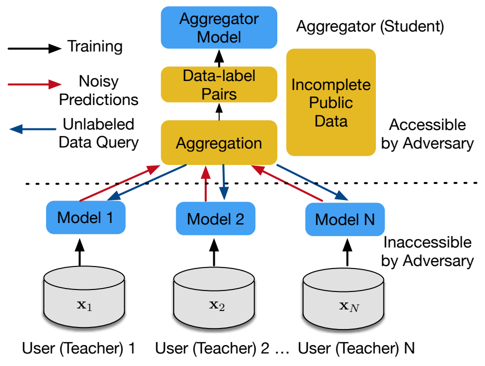


Fig. 1. Semi-supervised knowledge transfer framework.

learned on their private dataset by making predictions on those unlabeled instances. The aggregator collects answers from all users to label instances. Finally, the aggregator conducts semi-supervised learning on the collection of data-label pairs to train a joint model. Since the aggregator model is trained on the combination of users' query answers, it usually enjoys better generalization capability and higher accuracy.

B. Paillier Cryptosystem and DGK

We adopt Paillier cryptosystem as our additive homomorphic encryption tool for aggregating users' answers in a secure manner. We choose Paillier due to its efficiency in ciphertext size and in performing homomorphic operations. It has been proven its strong security guarantees. The limitation lies in that it only supports addition and multiplication operations over encrypted data. There are two types of keys involved: the public key pk is used to encrypt the plaintext m and the private key sk is used for decryption. The plaintext $m \in \mathbb{Z}_n$, where n is a large positive integer and $\mathbb{Z}_n$ is the set of integers modulo n. We denote the encryption of m with public key pk as $E_{pk}[m]$ and the decryption of ciphertext c using secret key sk as $E_{sk}[c]$. The homomorphic properties of the cryptosystem can be described as

$$E_{pk}[m_1 + m_2] = E_{pk}[m_1] \cdot E_{pk}[m_2], \tag{1}$$

$$E_{pk}[a \cdot m_1] = E_{pk}[m_1]^a, \tag{2}$$

where a is a constant that $a \in \mathbb{Z}_n$.

We also use DGK as our secure comparison tool for comparing each pair of aggregated values. DGK is built on homomorphic cryptosystem and is proven to be semantically secure. There are many variants of the protocol depending on whether the compared values are known to both parties, or whether the comparison result is kept secret. We adopt the most primitive one in our work: A has a private integer a, B has a private integer b and we want to compare a and b without revealing any party's value. In the protocol, a and b are compared by their binary representation in an interactive way.

C. Rényi Differential Privacy

Rényi Differential Privacy (RDP) [18] is proposed as a relaxation of the pure differential privacy ($\delta = 0$). It has natural advantages over the (ϵ, δ)-differential privacy since it composes nicely, and captures the privacy guarantee of Gaussian noise in a much cleaner way [2]. The concept is defined based on Rényi divergence:

Definition 1. *(Rényi divergence). The Rényi divergence of order $\alpha > 1$ between two distributions P and Q is defined as*

$$D_\alpha(P||Q) \triangleq \frac{1}{\alpha - 1} \log \mathbb{E}_{x\sim Q}\Big[\Big(\frac{P(x)}{Q(x)}\Big)^\alpha\Big].$$

Hence, Rényi Differential Privacy is defined as follows:

Definition 2. *((α, ϵ)-RDP). A randomized mechanism $\mathcal{M}$ guarantees (α, ϵ)-RDP with $\alpha \geq 1$ if for any adjacent inputs I and I', and any output set $O \subseteq \mathcal{O}$,*

$$D_\alpha(\mathcal{M}(I)||\mathcal{M}(I')) = \frac{1}{\alpha - 1} \log \mathbb{E}_{x\sim\mathcal{M}(I')}\Big[\Big(\frac{\Pr[\mathcal{M}(I) \in O]}{\Pr[\mathcal{M}(I') \in O]}\Big)^\alpha\Big] \leq \epsilon.$$

Similar to the definition of differential privacy, ϵ-RDP implies the ability of an adversary to inflict damage to any set of users is independent of whether any individual opts into, or out of the dataset. The RDP inherits several important properties from the standard definition of differential privacy, for example, whereas ϵ-differential privacy mechanism does not move the relative probabilities assigned to any two adjacent inputs by more than e^ϵ, ϵ-RDP mechanism does not move the relative probabilities by more than e^ϵ in expectation.

By the probability preservation proposition [18], if a mechanism $\mathcal{M}$ is (α, ϵ)-RDP, then we have:

$$e^{-\epsilon} \Pr[\mathcal{M}(I') \in O]^{\alpha/(\alpha-1)} \leq \Pr[\mathcal{M}(I) \in O] \leq (e^\epsilon \Pr[\mathcal{M}(I') \in O])^{\alpha/(\alpha-1)}. \tag{3}$$

Different mechanisms such as randomized response, Laplace and Gaussian noise addition can be applied to realize ϵ-RDP. Take the Gaussian mechanism for example:

Theorem 1. *(Gaussian noise addition. Corollary 3 of [18].) If f has sensitivity Δ, the Gaussian mechanism $\mathcal{M}$ such that*

$$\mathcal{M}[f(I)] = f(I) + \mathcal{N}(0, \sigma^2)$$

satisfies $(\alpha, \alpha\Delta^2/(2\sigma^2))$-RDP, and $\mathcal{N}(0, \sigma^2)$ is a normally distributed random variable with standard deviation σ^2 and mean 0.

If multiple RDP mechanisms are applied, it is equivalent to applying one RDP scheme:

Theorem 2. *(Composition.) If $\mathcal{M}_1$ guarantees (α, ϵ_1)-RDP and $\mathcal{M}_2$ guarantees (α, ϵ_2)-RDP, then $(\mathcal{M}_1, \mathcal{M}_2)$ guarantees $(\alpha, \epsilon_1 + \epsilon_2)$-RDP.*

In the latter section, we will derive the differential privacy guarantee of our mechanism based on the RDP.

IV. A Private Consensus Protocol

In this section, we first point out problems with Aggregation of Teacher Ensembles, illustrate the technical intuition of our solution, and finally describe our differentially-private consensus protocol.

A. Aggregation of Teacher Ensembles

Assume we have a set of users $\mathcal{U}$ each of whom privately possesses a subset of data belonging to K classes. Each user independently trains a local model on their private dataset. The aggregator, possessing a set of public data without labels, randomly chooses a subset of its data $\mathbf{x}$ to query each user. User u responds with $\mathbf{c}^u(x)$ — a probabilistic vector or a one-hot encoding vector indicating the chosen label — to the aggregator. The aggregator then adds up all responses for each instance. For all users, if we use the superscript to represent the user and the subscript to denote the index of the label, the total votes of label i for instance x is:

$$c_i(x) = \mathbf{e}_i^{\mathsf{T}} \cdot \sum_{u \in \mathcal{U}} \mathbf{c}^u(x) = \sum_{u \in \mathcal{U}} c_i^u(x), \tag{4}$$

where $\mathbf{e}_i$ represents a vector with the i-th entry being one and the rest being zeros, and $c_i^u(x)$ represents the prediction result of user u to the i-th class for x. For ease of presentation, we define the label with the highest votes for x as

$$i^* \triangleq \arg\max_i \{c_i(x)\}, \forall i \in \{1, ..., K\}. \tag{5}$$

Given an instance x and a pre-defined threshold T, we need to find out i^* as well as the number of votes $c_{i^*}(x)$ to see if the highest votes exceeds T. i^* is returned only when a plurality of users agree on the decision. The algorithm is summarized in Alg. 1. By running Alg. 1 [1] for each instance in $\mathbf{x}$,

Algorithm 1 Aggregation of Teacher Ensembles

```
Input: User u's prediction for the instance x: $c_i^u(x), \forall i \in \{1, ..., K\}$. A chosen threshold $T$.
1: if $c_{i^*}(x) \geq T$ then
2:     Return $i^*$.
3: else
4:     Return $\perp$.
5: end if
```

the aggregator obtains a number of data-label pairs, based on which it trains an aggregator model without direct access to each user's data.

The problem with secure sum and secure comparison protocols: We made the first attempt to aggregate each user's votes using a secure sum protocol. The aggregator is able to compute the sum without accessing each individual value, but the resulting sum has to be revealed for the aggregator to check if the plurality surpasses the threshold. In fact, only the label index has to be revealed and thus the aggregation result leaks more information than necessary. Similar to the reason above, if we apply a secure comparison protocol for checking the plurality, other label indices, apart from the one with the highest vote, are revealed as a result of comparison, even if none of the comparing values are released during the process.

Ideally, we expect a protocol which finds out whether the value of $c_{i^*}(x)$ surpasses T and i^*, yet without exposing $c_j(x), \forall j \in \{1, \ldots, K\}$, or any $j \neq i^*$ to ensure the minimal possible information is released.

B. Threat Model and Technical Intuition

Our protocol is designed against semi-honest users/servers and we assume the two servers S_1, S_2 would not collude with each other. This is a realistic assumption as the two servers may belong to different corporations and collusion would hurt their reputation. By the semi-honest setting, it is assumed that users and servers have common interests on the global model and thus would not tamper the data maliciously. Yet, it is possible for a subset of users, including one of the servers or not, to collude and intercept the private information of other honest users.

The foremost problem is to find out the ranking of different labels without learning anything about their aggregated sum. We convert the comparison of aggregated values into other forms. The symbol of a data instance x is omitted in discussion from now on for simplicity. To compare the aggregated value with T, we ask each user to randomly split its value as $c^u = a^u + b^u$, and submit $a^u - T/(2|\mathcal{U}|)$ to S_1 and $T/(2|\mathcal{U}|) - b^u$ to S_2. The comparison can instead be made between a^u and b^u:

$$\begin{aligned} \sum_{u \in \mathcal{U}} c^u \geq T &\Leftrightarrow \sum_{u \in \mathcal{U}} a^u + b^u \geq T/2 + T/2 \\ &\Leftrightarrow \sum_{u \in \mathcal{U}} a^u - T/(2|\mathcal{U}|) \geq \sum_{u \in \mathcal{U}} T/(2|\mathcal{U}|) - b^u. \end{aligned} \tag{6}$$

Similarly, to find out i^*, we have each user randomly split their values as $c_i^u = a_i^u + b_i^u, \forall i \in \{1, \ldots, K\}$, and send a_i^u to S_1 and b_i^u to S_2 for each i. The two servers respectively aggregate the values from users and compare $\sum_{u \in \mathcal{U}} a_i^u - a_j^u$ with $\sum_{u \in \mathcal{U}} b_j^u - b_i^u$ instead, since

$$\begin{aligned} c_i \geq c_j &\Leftrightarrow \sum_{u \in \mathcal{U}} a_i^u + b_i^u \geq \sum_{u \in \mathcal{U}} a_j^u + b_j^u \\ &\Leftrightarrow \sum_{u \in \mathcal{U}} a_i^u - a_j^u \geq \sum_{u \in \mathcal{U}} b_j^u - b_i^u. \end{aligned} \tag{7}$$

Note that *secure comparison* primitive can be adopted to compare two values owned by two parties without either party obtaining anything about the other.

The second issue is to prevent revealing the ranking of the rest labels $j \neq i^*$. We adopt a blind-and-permute sub-protocol to permute the sequences without revealing to any party the permutation order. In detail, S_1 and S_2 sequentially permute the same sequence without letting the other know about its permutation order. The resulting sequence would be permuted twice, but the order remains unknown to either server. Then, the two servers can compare each pair of elements in the sequence without knowing about the true indices.

C. A Blind-and-Permute Protocol

To prevent the indices from being revealed, we apply blind-and-permute sub-protocol as follows: Each server S privately generates a permutation $\pi_s(\cdot)$ on $\{1, \dots, K\}$ and randomized masks. The two servers sequentially performs permutation and masking to the original sequence such that the resulting sequence is permuted and distorted twice. Beyond that, sequences are encrypted by additively homomorphic encryption with the private key owned by only one of the servers. Our blind-and-permute protocol allows two servers to agree upon an unknown sequence order $\pi(\cdot) = \pi_1(\pi_2(\cdot))$. Please refer to Alg. 2 for a detailed description. To restore the sequence, the two servers sequentially revert the permutation and remove masks from the sequence. The restoration details are given in Alg. 3.

Algorithm 2 Blind-and-Permute

Setup: S_1 randomly chooses a permutation π_1, randomized masks $\mathbf{r}_1$, and a pair of additively homomorphic keys (pk_1, sk_1). S_2 randomly chooses a permutation π_2, randomized masks $\mathbf{r}_2, \mathbf{r}_3$, and a pair of additively homomorphic keys (pk_2, sk_2). All public keys are released by the PKI.

```
Input: $S_1$ has an encrypted sequence $E_{pk_2}[\mathbf{a}]$ and $S_2$ holds the sequence $E_{pk_1}[\mathbf{b}]$.
Output: $S_1$ holds the sequence $\pi(\mathbf{a}+\mathbf{r})$ and $S_2$ holds $\pi(\mathbf{b}+\mathbf{r})$ where $\pi(\cdot) = \pi_1(\pi_2(\cdot))$ and $\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2$.
1: $S_1$ adds encrypted random mask $E_{pk_2}[\mathbf{r}_1]$ to $E_{pk_2}[\mathbf{a}]$, and sends $E_{pk_2}[\mathbf{a} + \mathbf{r}_1]$ to $S_2$.
2: $S_2$ decrypts it, adds random mask $\mathbf{r}_2$, permutes the sequence with $\pi_2$. It sends $\pi_2(\mathbf{a} + \mathbf{r}_1 + \mathbf{r}_2)$ to $S_1$.
3: $S_1$ permutes the received sequence with $\pi_1$, and obtains $\pi_1(\pi_2(\mathbf{a} + \mathbf{r}_1 + \mathbf{r}_2))$. It sends encrypted random mask $E_{pk_1}[\mathbf{r}_1]$ to $S_2$.
4: $S_2$ adds up the encrypted values $E_{pk_1}[\mathbf{b}], E_{pk_1}[\mathbf{r}_1]$ and $E_{pk_1}[\mathbf{r}_2]$, permutes the results by $\pi_2$ and adds $E_{pk_1}[\mathbf{r}_3]$ to the permutation result. It sends $E_{pk_1}[\pi_2(\mathbf{b}+\mathbf{r}_1+\mathbf{r}_2)+\mathbf{r}_3]$ and $E_{pk_2}[-\mathbf{r}_3]$ to $S_1$.
5: $S_1$ decrypts the received sequence with $pk_1$ and then re-encrypts it with $pk_2$. It then adds $E_{pk_2}[-\mathbf{r}_3]$ to the result to obtain $E_{pk_2}[\pi_2(\mathbf{b} + \mathbf{r}_1 + \mathbf{r}_2)]$. Permuting the sequence with $\pi_1$, $S_1$ sends permuted and encrypted sequence to $S_2$.
6: $S_2$ decrypts the received sequence with $pk_2$ and obtains $\pi_1(\pi_2(\mathbf{b} + \mathbf{r}_1 + \mathbf{r}_2))$.
```

By the secure comparison primitive, S_1 and S_2 can compare the aggregated votes of every pair of labels to obtain the one with the highest votes, or to check if the votes exceed a threshold, unaware of the true label index. To recover the true label, the two servers need to run the restoration protocol. At the core, each server reverts its permutation and removes its mask from the sequence respectively. Alg. 3 provides the detail and note that $\pi_s^{-1}(\cdot)$ denotes the reversion of the permutation $\pi_s(\cdot)$.

Algorithm 3 Restoration

Setup: S_1 randomly chooses a randomized mask $\mathbf{r}_1$, and a pair of additively homomorphic keys (pk_1, sk_1). S_2 randomly chooses a randomized mask $\mathbf{r}_2$, and a pair of additively homomorphic keys (pk_2, sk_2). All public keys are released by the PKI.

```
Input: A given index $\pi(i^*)$.
Output: $i^*$.
1: $S_2$ encrypts the sequence $\pi(\mathbf{e}_{i^*})$ with $pk_2$ and sends $E_{pk_2}[\pi(\mathbf{e}_{i^*})]$ to $S_1$.
2: $S_1$ reverts its permutation and adds an encrypted randomized mask $E_{pk_2}[\mathbf{r}_1]$: $\pi_1^{-1}(E_{pk_2}[\pi(\mathbf{e}_{i^*})])E_{pk_2}[\mathbf{r}_1] = E_{pk_2}[\pi_2(\mathbf{e}_{i^*}) + \mathbf{r}_1]$. It sends the result to $S_2$.
3: $S_2$ decrypts the result and sends $\pi_2(\mathbf{e}_{i^*}) + \mathbf{r}_1$ to $S_1$.
4: $S_1$ subtracts $\mathbf{r}_1$ from the result and encrypts it with $pk_1$. It sends $E_{pk_1}[\pi_2(\mathbf{e}_{i^*})]$ to $S_2$.
5: $S_2$ reverts the permutation and adds a randomized mask $\mathbf{r}_2$: $\pi_2^{-1}(E_{pk_1}[\pi_2(\mathbf{e}_{i^*})])E_{pk_1}[\mathbf{r}_2]$. It sends $E_{pk_1}[\mathbf{e}_{i^*} + \mathbf{r}_2]$ to $S_1$.
6: $S_1$ decrypts the result and returns it to $S_2$.
7: $S_2$ removes $\mathbf{r}_2$ from the result and obtain $\mathbf{e}_{i^*}$.
```

D. Ensuring Differentially Privacy

With the intuition in Sec. IV-B and the protocols in Sec. IV-C, we are able to construct an instance of the aggregation of teacher ensembles without revealing any information except for the final output to any party. However, even with the final output, it is likely for an adversary to infer properties of the training data. Differential privacy was proposed to resolve the issue by randomizing the output. Specifically, the private aggregation of teacher ensembles [2] is as follows:

Algorithm 4 Private Aggregation of Teacher Ensembles

```
Input: User $u$'s prediction for the instance $x$: $c_i^u(x), \forall i \in \{1, ..., K\}$. A chosen threshold $T$.
1: if $c_{i^*}(x) + \mathcal{N}(0, \sigma_1^2) \geq T$ then
2:     Return $\tilde{i}^* \triangleq \arg\max_i \{c_i(x) + \mathcal{N}(0, \sigma_2^2)\}$.
3: else
4:     Return $\perp$.
5: end if
```

Alg. 4 comprises an instance of the Sparse Vector Technique and an instance of the Report Noisy Maximum [19]. For a given threshold T and a given instance x, we first find the label with the highest votes $c_{i^*}(x)$ and add noise to it. The aggregator compares the highest noisy votes with the threshold: if it surpasses T, we consider the majority of users pick the same label for x, otherwise no consensus is achieved. If consensus has been achieved, the aggregator returns the maximum of the noisy votes. Note that $\tilde{i}^*$ is different from i^* since the former represents the label of the highest noisy votes and the latter is the label of the highest votes.

However, it remains a problem to add additional noise without revealing any intermediate result. If it depends on the aggregator to perform the perturbation, the aggregator would learn the exact amount of noise, risking revealing the true intermediate result. Hence, our scheme utilizes the additive property of Gaussian noise and lets each user locally generate the randomized noise. Let z_1^u denote an instance of $\mathcal{N}(0, \sigma_1^2/(2|\mathcal{U}|))$ and z_2^u be an instance of $\mathcal{N}(0, \sigma_2^2/(2|\mathcal{U}|))$ generated by user u locally. If the random noise from all users are aggregated, we have respectively $z_1 \triangleq \sum_{u \in U} z_1^u \sim \mathcal{N}(0, \sigma_1^2/2)$ and $z_2 \triangleq \sum_{u \in U} z_2^u \sim \mathcal{N}(0, \sigma_2^2/2)$.

The challenge lies in that the protocol should only reveal $\tilde{i}^*$ but not i^* to avoid spending additional privacy budget on the releasing of i^*. Our solution is to enforce the same permutation order on the original votes and noisy votes in the threshold checking. The detail of the process is illustrated in Alg. 5. Overall, the algorithm consists of three parts which we will give an overview as follows.

Algorithm 5 Private Consensus Protocol

```
Input: Each user's prediction for instance $x$: $\mathbf{c}^u(x), \forall u \in \mathcal{U}$.
    Privacy parameters $\sigma_1$ and $\sigma_2$.
Output: Label with the highest noisy vote.
1: Setup: User $u \in \mathcal{U}$ randomly splits its prediction as $\mathbf{c}^u(x) = \mathbf{a}^u(x) + \mathbf{b}^u(x)$. Each user also generates randomized noise $\mathbf{z}_1^u$ and $\mathbf{z}_2^u$.
2: Secure Sum: User $u \in \mathcal{U}$ sends $E_{pk_2}[\mathbf{a}^u]$, $E_{pk_2}[\mathbf{a}^u - (T/2|\mathcal{U}|)\mathbf{I} + \mathbf{z}_1^u]$ to $S_1$ and $E_{pk_1}[\mathbf{b}^u]$, $E_{pk_1}[(T/2|\mathcal{U}|)\mathbf{I} - \mathbf{b}^u - \mathbf{z}_1^u]$ to $S_2$.
   $S_1$ aggregates the value from all users in $\mathcal{U}$: $E_{pk_2}[\mathbf{a}] = \prod_{u \in \mathcal{U}} E_{pk_2}[\mathbf{a}^u]$ and $E_{pk_2}[\mathbf{a} - (T/2)\mathbf{I} + \mathbf{z}_1] = \prod_{u \in \mathcal{U}} E_{pk_2}[\mathbf{a}^u - (T/2|\mathcal{U}|)\mathbf{I} + \mathbf{z}_1^u]$.
   $S_2$ aggregates the value from all users in $\mathcal{U}$: $E_{pk_1}[\mathbf{b}] = \prod_{u \in \mathcal{U}} E_{pk_1}[\mathbf{b}^u]$ and $E_{pk_1}[(T/2)\mathbf{I} - \mathbf{b} - \mathbf{z}_1] = \prod_{u \in \mathcal{U}} E_{pk_1}[(T/2|\mathcal{U}|)\mathbf{I} - \mathbf{b}^u - \mathbf{z}_1^u]$.
3: Blind-and-Permute: $S_1$ and $S_2$ execute Alg. 2. $S_1$ obtains $\pi(\mathbf{a}+\mathbf{r})$ and $\pi(\mathbf{a} - (T/2)\mathbf{I} + \mathbf{z}_1 + \mathbf{r}')$. $S_2$ obtains $\pi(\mathbf{b}+\mathbf{r})$ and $\pi((T/2)\mathbf{I} - \mathbf{b} - \mathbf{z}_1 + \mathbf{r}')$.
4: Secure Comparison: for each pair of $i, j \in \{1, \ldots, K\}$, by using DGK comparison protocol and Eqn. (7), $S_1$ and $S_2$ compare $\pi(\mathbf{a}+\mathbf{r})$ with $\pi(\mathbf{b}+\mathbf{r})$ to find $\pi(i^*)$.
5: Threshold Checking: by using DGK and Eqn. (6), $S_1$ and $S_2$ compare $(\mathbf{a} - (T/2)\mathbf{I} + \mathbf{z}_1 + \mathbf{r}')_{\pi(i^*)}$ with $((T/2)\mathbf{I} - \mathbf{b} - \mathbf{z}_1 + \mathbf{r}')_{\pi(i^*)}$ to decide if $c_{\pi(i^*)} + \mathcal{N}(0, \sigma_1^2) \geq T$. If not, the protocol returns $\perp$.
6: Secure Sum: User $u \in \mathcal{U}$ sends $E_{pk_2}[\mathbf{a}^u + \mathbf{z}_2^u]$ to $S_1$ and $E_{pk_1}[\mathbf{b}^u + \mathbf{z}_2^u]$ to $S_2$. $S_1$ aggregates the value from all users in $\mathcal{U}$: $E_{pk_2}[\mathbf{a} + \mathbf{z}_2] = \prod_{u \in \mathcal{U}} E_{pk_2}[\mathbf{a}^u + \mathbf{z}_2^u]$. $S_2$ aggregates the value from all users in $\mathcal{U}$: $E_{pk_1}[\mathbf{b} + \mathbf{z}_2] = \prod_{u \in \mathcal{U}} E_{pk_1}[\mathbf{b}^u + \mathbf{z}_2^u]$.
7: Blind-and-Permute: $S_1$ and $S_2$ execute Alg. 2 to respectively obtain $\pi'(\mathbf{a}+\mathbf{r})$ and $\pi'(\mathbf{b}+\mathbf{r})$.
8: Secure Comparison: for each pair of $i, j \in \{1, \ldots, K\}$, by using DGK and Eqn. (7), $S_1$ and $S_2$ compare $\pi'(\mathbf{a}+\mathbf{r})$ with $\pi'(\mathbf{b}+\mathbf{r})$ to find $\pi'(\tilde{i}^*)$.
9: Restoration: $S_1$ and $S_2$ execute Alg. 3 to return $\tilde{i}^*$.
```

In the first part (line 1 to 4 in Alg. 5), each user randomly splits each of its prediction result into two shares, sending one share to S_1 and the other to S_2. The two servers respectively aggregate the partial values from all users and runs Blind-and-Permute on the aggregated values. By permutation, S_1 and S_2 respectively obtain $\pi(\mathbf{a}+\mathbf{r})$ and $\pi(\mathbf{b}+\mathbf{r})$. As the two sequences share the same permutation order and a common bias, S_1 and S_2 can find out $\pi(i^*)$. Note that we do not restore the permutation order in this part.

In the second part (line 5 in Alg. 5), we test if the randomized highest votes is larger than a threshold. The key is that the differentially-private sequence shares the same permutation order π which allows threshold checking without learning the label with the highest votes. If threshold checking fails, no consensus is achieved for x and thus the instance is discarded.

In the third part (line 7 to 9 in Alg. 5), servers conduct Blind-and-Permute and Restoration protocols to find the label with the highest noisy vote $\tilde{i}^*$ and include $(x, \tilde{i}^*)$ as a data-label pair for training.

V. Security and Privacy Analysis

We give the security proof of our protocol and the differential privacy guarantee.

A. Security

Theorem 3. ***(Correctness)*** *Alg. 5 returns $\tilde{i}^*$ if $c_{i^*}(x) + \mathcal{N}(0, \sigma_1^2) \geq T$. Otherwise, a stop string is returned.*

The proof is straightforward from the algorithm and thus are omitted. Then we show that our protocol is privacy-preserving in the semi-honest setting with non-colluding servers. The sketch proof is provided.

Theorem 4. ***(Privacy)*** *If servers are semi-honest and non-colluding, there is a simulator* SIM *of Alg. 5 such that for any $\mathcal{V} \subseteq \mathcal{U} \bigcup \{S_s\}, s \in \{1, 2\}$, the output of* SIM *is computationally indistinguishable from the output of* $\mathrm{REAL}_{\mathcal{V}}^{\mathcal{U}}$*:*

$$\mathrm{REAL}_{\mathcal{V}}^{\mathcal{U}}(\mathbf{c}^{\mathcal{V}}) \approx_c \mathrm{SIM}_{\mathcal{V}}^{\mathcal{U}}(\mathbf{c}^{\mathcal{V}}),$$

where $\mathbf{c}^{\mathcal{V}}$ is defined as

$$\mathbf{c}^{\mathcal{V}} = \{\mathbf{c}^u | u \in \mathcal{V}\}.$$

Proof. (Sketch) We do not repeat the proofs of DGK comparison protocol and secure sum protocol, but focus on the proof of the Blind-and-Permute and Restoration protocol.

The main point is to prove that the joint view of the parties in $\mathcal{V}$ does not depend on the inputs of the parties not in $\mathcal{V}$. The simulator can produce a simulation by running the semi-honest users on their true inputs, and all other users on dummy inputs. In the Blind-and-Permute protocol, we discuss two cases. If $S_1 \in \mathcal{V}$, the joint view of $\mathcal{V}$ is $\{\mathbf{a}^u, \mathbf{b}^u | u \in \mathcal{V}\} \bigcup \{\mathbf{r}_1, E_{pk_2}[\mathbf{a}], \pi_2(\mathbf{a} + \mathbf{r}_1 + \mathbf{r}_2), \pi_2(\mathbf{b} + \mathbf{r}_1 + \mathbf{r}_2) + \mathbf{r}_3, E_{pk_2}[-\mathbf{r}_3]\}$. Since $\pi_2, \mathbf{r}_2, \mathbf{r}_3$ are randomly generated and unknown to users in $\mathcal{V}$, $\mathbf{a}$ and $\mathbf{b}$ can be replaced with randomly generated permutation and numbers. The encryptions of $\mathbf{a}$ can be replaced by the encryptions of 0 (padded to the appropriate length). Due to IND-CPA security, the joint view of users in $\mathcal{V}$ will be identical to that in $\mathrm{REAL}_{\mathcal{V}}^{\mathcal{U}}$.

Likewise, if $S_2 \in \mathcal{V}$, the joint view of $\mathcal{V}$ is $\{\mathbf{a}^u, \mathbf{b}^u | u \in \mathcal{V}\} \bigcup \{E_{pk_1}[\mathbf{b}], \mathbf{a}+\mathbf{r}_1, \mathbf{r}_2, E_{pk_1}[\mathbf{r}_1], \mathbf{r}_3, \pi_1(\mathbf{b}+\mathbf{r}_1+\mathbf{r}_2)\}$. Since $\pi_1, \mathbf{r}_1$ are randomly generated and unknown to users in $\mathcal{V}$, $\mathbf{a}$ and $\mathbf{b}$ can be replaced with randomly generated permutation and numbers. The encryptions of $\mathbf{b}$ can be replaced by the encryptions of 0. Due to IND-CPA security, the joint view of users in $\mathcal{V}$ will be identical to that in $\mathrm{REAL}_{\mathcal{V}}^{\mathcal{U}}$. Hence, no additional information about users in $\mathcal{U} \backslash \mathcal{V}$ is revealed in executing Blind-and-Permute.

The proof of the restoration protocol is similar to that of Blind-and-Permute. If $S_1 \in \mathcal{V}$, the joint view of $\mathcal{V}$ is $\{\mathbf{a}^u, \mathbf{b}^u | u \in \mathcal{V}\} \bigcup \{E_{pk_2}[\pi(\mathbf{e}_{i^*})], \pi_2(\mathbf{e}_{i^*}), \mathbf{e}_{i^*} + \mathbf{r}_2\}$. Since $\pi_2, \mathbf{r}_2$ are unknown to $\mathcal{V}$, one can replace $\mathbf{e}_{i^*}$ with any random sequence and the joint view of $\mathcal{V}$ would be identical to that in $\mathrm{REAL}_{\mathcal{V}}^{\mathcal{U}}$. If $S_1 \in \mathcal{V}$, the joint view of $\mathcal{V}$ is $\{\mathbf{a}^u, \mathbf{b}^u | u \in \mathcal{V}\} \bigcup \{\pi_2(\mathbf{e}_{i^*}) + \mathbf{r}_1, E_{pk_1}[\mathbf{e}_{i^*}]\}$. For similar reasons as above, the joint view of $\mathcal{V}$ would be identical to that in $\mathrm{REAL}_{\mathcal{V}}^{\mathcal{U}}$.

Finally, the security proof of our private consensus protocol is built on the security proofs of DGK, secure sum, Blind-and-Permute and Restoration protocols. □

B. Differential Privacy

In this section, we analyze the differential privacy guarantee of the final output. As a result, we have that

Theorem 5. *For any $\delta \in (0,1)$, Alg. 5 guarantees $\left(\sqrt{2\left(9/\sigma_1^2 + 2/\sigma_2^2\right)\log 1/\delta} + \left(9/2\sigma_1^2 + 1/\sigma_2^2\right), \delta\right)$-differential privacy.*

Alg. 5 comprises two mechanisms: line 2 to 5 is an instance of the Sparse Vector Technique and line 6 to 9 is an instance of the Report Noisy Maximum.

Lemma 1. *Sparse Vector Technique satisfies $\left(\alpha, 9\alpha/2\sigma_1^2\right)$-RDP.*

Proof. The proof is based on the proof of Theorem 3.23 in [19]. We define the noisy threshold as $\hat{T}$ and the perturbation noise of the k queried instances before the one exceeding the threshold is found as $\{\nu_m\}_{m=1}^k$. It means the last instance in the sequence is chosen, but instances 1 to $k-1$ are not chosen. We restart the mechanism whenever one above the threshold is found. We fix the values of $\nu_1, \ldots, \nu_{k-1}$ and take probabilities over the randomness of ν_k and $\hat{T}$. Owing to the additive property of the Gaussian noise, we can rewrite the threshold comparison of the instance m as:

$$\begin{aligned}&c_{i^*,m} + \mathcal{N}(0, \sigma_1^2) \geq T \Leftrightarrow \\ &c_{i^*,m} + \mathcal{N}(0, \sigma_1^2/p) \geq T + \mathcal{N}(0, \sigma_1^2/q), \\ &\text{s.t. } 1/p + 1/q = 1.\end{aligned}$$

We define the highest noisy votes of instance 1 to $k-1$ as $g(I) = \max_{m<k}(c_{i^*,m}(I) + \nu_m)$. The probability that the k-th instance is above the threshold is

$$\Pr_{\hat{T},\nu_k}\left[\hat{T} > g(I) \text{ and } c_{i^*,k}(I) + \nu_k \geq \hat{T}\right] =$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \Pr[\nu_k = \nu]\Pr[\hat{T} = t]\mathbb{1}[t \in (g(I), c_{i^*,k}(I) + \nu)] d\nu dt.$$

We make a change of variables by defining

$$\hat{\nu} = \nu + g(I) - g(I') + c_{i^*,k}(I') - c_{i^*,k}(I),$$
$$\hat{t} = t + g(I) - g(I'),$$

and note that for any I, I', $\|\nu - \hat{\nu}\|_2 = |\nu - \hat{\nu}| \leq 2$, and $\|\hat{t} - t\|_2 = |\hat{t} - t| \leq 1$. Hence by Eqn. (3), it can be obtained that

$$\Pr[\nu_k = \hat{\nu}] \leq \left[e^{\epsilon_p}\Pr[\nu_k = \nu]\right]^{\frac{\alpha-1}{\alpha}}, \epsilon_p = 2p\alpha/\sigma_1^2$$
$$\Pr[\hat{T} = \hat{t}] \leq \left[e^{\epsilon_q}\Pr[\hat{T} = t]\right]^{\frac{\alpha-1}{\alpha}}, \epsilon_q = q\alpha/2\sigma_1^2.$$

By the composition theorem of RDP, the Sparse Vector Technique satisfies $(\alpha, \epsilon_p + \epsilon_q)$-RDP. If we minimize the value of $\epsilon_p + \epsilon_q$ over the constraint $1/p + 1/q = 1$, we obtain $\epsilon_p + \epsilon_q = 9\alpha/2\sigma_1^2$. □

Lemma 2. *Report Noisy Maximum satisfies $\left(\alpha, \alpha/\sigma_2^2\right)$-RDP.*

Proof. The result can be derived as one entry difference would change the maximum counts by at most 1. □

Proof. (Thm. 5) By Lemma 1, 2 and Thm. 2, we can prove Alg. 5 satisfies $\left(\alpha, 9\alpha/2\sigma_1^2 + \alpha/\sigma_2^2\right)$-RDP. By Theorem 5 of [2], it can be easily proved that Alg. 5 guarantees (ϵ, δ)-differential privacy with

$$\epsilon \geq \sqrt{2\left(9/\sigma_1^2 + 2/\sigma_2^2\right)\log 1/\delta} + \left(9/2\sigma_1^2 + 1/\sigma_2^2\right),$$

and the equality holds when $\alpha = 1 + \sqrt{\frac{2\log 1/\delta}{9/\sigma_1^2 + 2/\sigma_2^2}}$. □

VI. Evaluation

In this section, we discuss the implementation detail of our private consensus protocol, its computational and storage overhead, as well as the accuracy and privacy performance.

A. Implementation

A prototype of our protocol is built on the additive homomorphic cryptosystem Paillier, secure comparison primitive DGK, and the `Pytorch` library. The Paillier crypto primitive has a key size of 64 bit. The `Pytorch` library is used as the framework for distributed machine learning. Unfortunately, we faced a series of challenges in adopting the primitives in our implementation. The following shows how we overcome these challenges.

Extended supports to float numbers. Since the original Paillier primitive only supports integer encryption, we perform a conversion from float numbers to integers while retaining a certain precision. More precisely, assuming the object is a float number R, we cut off the fractional part which is less than $2^{-16} \approx 1.526 \times 10^{-5}$ from R to retain its sign bit and the most significant 31 bit. Then we transform R into a 32 bit positive integer R^I by

$$R^I = R \cdot 2^{16} + 2^{31}, \ \forall R \in [-2^{15}, 2^{15}). \quad (8)$$

Although a higher precision can be obtained, the additional bits would cause extra burden since the protocol involves bitwise operation.

Encrypted numbers converted to tensors. We use `send` and `recv` interfaces in `torch.distributed` for the communication between the user and server, as well as between the two servers. The pair of interfaces support synchronous communication between processes by inter-process communication or between devices by TCP. Since tensor objects are passed around by the two interfaces, we convert the numbers into tensors. Plaintext can be easily transformed into tensors but the encrypted number, due to the ciphertext expansion, cannot fit within the capacity of a single tensor object. Hence we develop segmentation and recomposition interfaces added upon the `send` and `recv`: before being sent, the ciphertext is divided into units with each unit being a 18-digit long decimal number which could fit into a tensor, and the ciphertext is sent by segments; upon receiving these segments, we re-compose the original ciphertext from tensors.

Encrypt numbers efficiently. Since the aggregator needs to collect a sufficiently large number of instances to train its model, the encryption primitive has to be applied to each instance, which makes the implementation highly inefficiently. Thus we first propose a countermeasure to split instances into batches and run encryptions in parallel. However, in the first place, we curiously found that the process does not gain any speedup as expected and acts as if all encryptions are done sequentially. We spot the bottleneck that almost all encryptions require random number generation which relies on a common generator, but the generator is not sufficiently fast in generating random numbers. Thus we made a tweak by generating a table of random numbers beforehand and have each process acquire a random number with the current time as the index to access the table. The approach finally effectively parallelize the encryption process and gains significant speedup.

B. Computational and Communication Costs

In this section, we show the computational and communication costs evaluated by experiments. Our protocol is tested on a server with Intel Xeon CPU E5-2650 v3. Each result is evaluated on 1000 instances from 10 classes, and is averaged over 755 rounds. We record the averaged per-step running time as a way to gauge the computational cost, the result of which is given in Table I. From the results, we found step (4)(5)(8) are dominant in the overall running time costs, owing to the secure comparison operation of DGK involves bitwise encryption and multiplication. If faster secure comparison primitives can be adopted, it would speed up the entire execution of the protocol.

TABLE I
COMPUTATIONAL COSTS

Step	Average Running Time (s)
Blind-and-permute (3)	4.972
Secure Comparison (4)	745.163
Threshold Checking (5)	475.962
Blind-and-permute (7)	1.847
Secure Comparison (8)	848.585
Restoration (9)	2.985
Overall	2079.514

We also measured the communication overhead by the size of the message passed between different parties. The message are stored in `Numpy` format. The results are evaluated on 1000 instances of 10 classes over 755 rounds as well. From Table II, we can tell that there is a light-weighted communication overhead on the user. The message transferred in the Blind-and-Permute and Restoration protocol is approximately 3 times larger than the plaintext, due to the ciphertext expansion of the Paillier crypto scheme. Overall, the message transferred between servers reaches the largest in the Secure Comparison step, which is around 4.5 times larger than the Threshold Checking step, since the former performs comparison for each pair of classes. Note that for server-to-server, the number given in the table is the size of the message exchanged between the two servers. It is reasonable that the message size in step(4)(5)(8) is relatively large because each number under comparison is encrypted bit-by-bit.

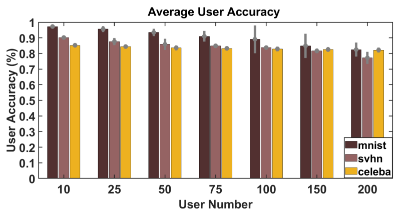
TABLE II
COMMUNICATION COSTS

Step	Message Size Per Party (KB)
Secure Sum (2)	234 (user-to-server)
Blind-and-Permute (3)	469 (server-to-server)
Secure Comparison (4)	33750 (server-to-server)
Threshold Checking (5)	7500 (server-to-server)
Secure Sum (6)	117 (user-to-server)
Blind-and-Permute (7)	234 (server-to-server)
Secure Comparison (8)	33750 (server-to-server)
Restoration (9)	234 (server-to-server)

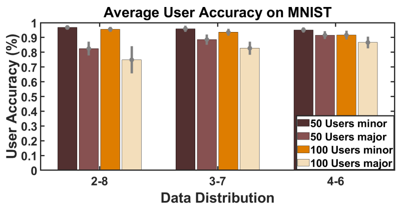
C. Accuracy and Privacy

We tested our private consensus protocol under a variety of data distribution and privacy settings on three datasets: MNIST, SVHN, and CelebA. The handwritten digits dataset MNIST contains a training set of 60000 examples and a testing set of 10000 examples. The real-world digits dataset SVHN contains over 60000 training examples and over 20000 testing examples. The face attributes datasets CelebA consists of 200000 images, each with 40 binary attributes. The aggregator's model is built on the Inception V3 neural network. To evaluate the performance, we adopt two metrics: *label accuracy*, which shows the percentage of the aggregator's instances labeled correctly by the consensus protocol; and *aggregator accuracy*, which is the aggregator's accuracy by training on the aggregation results. The metrics are measured under a variety of privacy levels as well as different data distributions. For each dataset, we set aside 9000 training samples for the aggregator, and the rest is distributed across the users. A threshold of 60% is chosen by default, meaning that consensus is achieved only when over 60% of the total number of users agree on the labels.

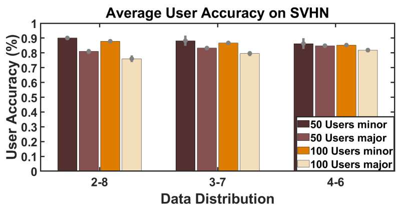
We first measure *user accuracy*, *i.e.,* the average accuracy that a user achieves on its local data, under a variety of data distributions. Fig. 2(a) shows the results when the training data is evenly distributed across all users. It is obvious that, as the number of users increases, the size of local dataset decreases



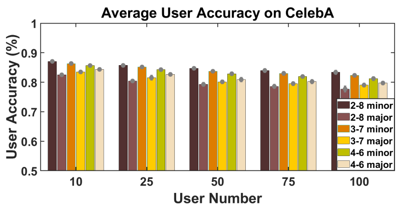
(a)



(b)



(c)



(d)

Fig. 2. (a) User accuracy when data is evenly distributed: average user accuracy gets lower with the increase of user numbers. (b)(c)(d) show that when data is unevenly distributed, the more unbalanced the dataset, the higher fluctuation of the user accuracy.

and thus the average user accuracy gets lower. Fig. 2(b), 2(c), and 2(d) show the average user accuracy when data is unevenly distributed across users. Three data distributions are chosen, namely division 2-8, 3-7 and 4-6. Division 2-8 represents that 20% of the data is held by 80% of the users, and the rest should be interpreted in a similar way. For fair evaluation, we do not show the average user accuracy, but the accuracies of the majority of users (80% of the users who holds 20% of the data), and minority of users (20% of the users who holds 80% of the data). Almost for all datasets, when the data is distributed more unevenly, there is a larger gap between the two, indicating that the accuracies are highly fluctuated across users. In the following, without particular statements, the data is evenly distributed across users by default.

To show the effectiveness of our private consensus protocol, we compare it with a naive *baseline* where the aggregator simply aggregates all noisy votes and pick the highest one as the label. For fair comparison, we apply the same differential privacy scheme and the same privacy level. Fig. 3(a) and Fig. 3(c) show the label accuracy for MNIST and SVHN. The results show that our private consensus protocol almost always improves accuracy performance compared with the baseline for evenly distributed data. Given the labeled instances, the semi-supervised trained aggregator also achieves higher accuracies across all privacy settings, as shown in Fig. 3(b) and Fig. 3(d). An exception is that when the number of users is small, for example, 25, our consensus protocol performs slightly worse. As we analyze, it is because when the user number is small, Threshold Checking discards votes that could have positively contributed to aggregation results, resulting in a limited number of instances that the aggregator trains on, and thus a degraded accuracy performance overall.

From the general trend of the figures, we can see that for all settings, the label and aggregator accuracy is higher as the privacy level gets lower. But the aggregator accuracy is not necessarily higher as the number of users grows. For the baselines, the aggregator accuracy always decreases with the number of users, but not for our consensus based method. This is because that although a larger group of users lead to a lower average user accuracy (Fig. 2(a)), our method effectively filters out invalid instances to improve the performance of the aggregator model.

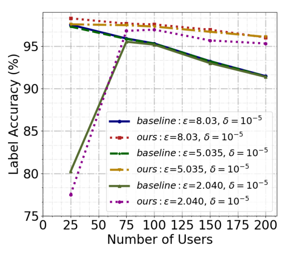
TABLE III
PROPORTION OF RETAINED SAMPLES/ LABEL ACCURACY (SVHN)

No. of Users	2-8	3-7	4-6
10	0.561 /0.746	0.578/0.745	0.586/0.761
25	0.691 / 0.921	0.707/0.924	0.707/0.925
50	0.802 / 0.933	0.817/0.935	0.833/0.934
75	0.823/0.935	0.855 / 0.934	0.871/0.933
100	0.816/0.937	0.858/0.931	0.877/0.931

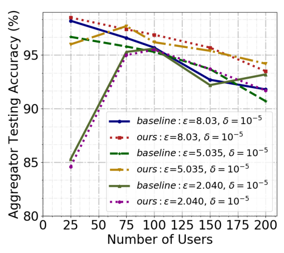
We also measured the accuracy when different types of labels are collected. One-hot stands for the binary vote where all entries are 0 except for the label selected. Softmax represents the probabilistic labels that the softmax layer outputs. Contrary to our assumption that the probabilistic labels would provide more information, the aggregator model yields a lower accuracy on softmax labels, as shown in Fig. 4(a), 4(b), 4(c) and 3(d). That essentially indicates the aggregation of softmax outputs do not necessarily contain more useful information. Hence one-hot labels are sufficient in the majority voting setting.

We also compared the accuracy result with varying voting thresholds from 30% to 90% of the total number of users, by setting a fixed privacy level ($\epsilon = 8.19, \delta = 10^{-6}$). As the results in Fig. 5(a) and Fig. 5(b) show, the aggregator accuracy is not the highest when the threshold is the lowest (30%) or highest (90%), but in the middle (around 60% to 70%), and the highest accuracy point also differs by the total number of users. The result naturally supports the majority voting scheme and provides reference for properly choosing the threshold.

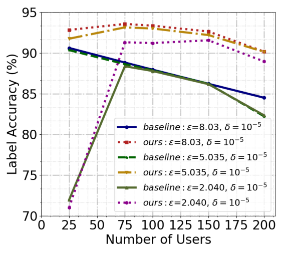
Fig. 5(c) and 5(d) show the aggregator accuracies when the



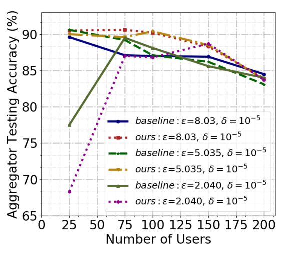
(a) Comparison of label accuracy (MNIST).



(b) Comparison of aggregator accuracy (MNIST).

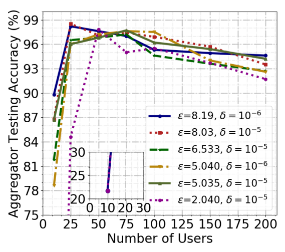


(c) Comparison of label accuracy (SVHN).

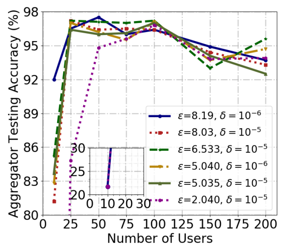


(d) Comparison of aggregator accuracy (SVHN).

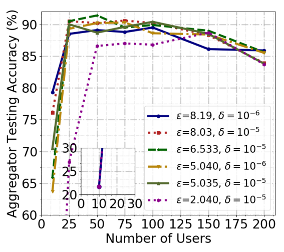
Fig. 3. Label accuracy and aggregator accuracy of MNIST and SVHN when data is evenly distributed across users.



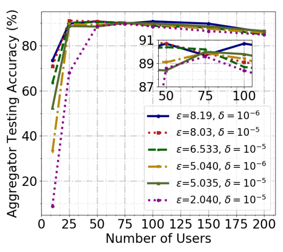
(a) Aggregator accuracy with one-hot labels (MNIST).



(b) Aggregator accuracy with softmax labels (MNIST).

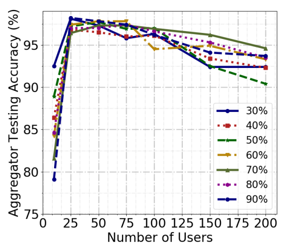


(c) Aggregator accuracy with one-hot labels (SVHN).

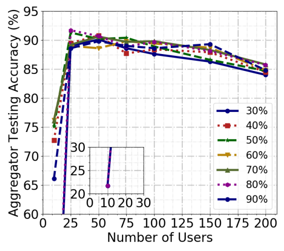


(d) Aggregator accuracy with softmax labels (SVHN).

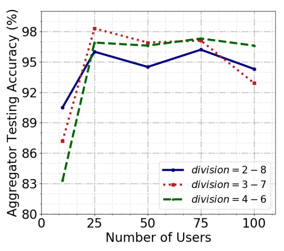
Fig. 4. Aggregator accuracy of MNIST and SVHN when the labels are one-hot and softmax: softmax is no better than one-hot labels.



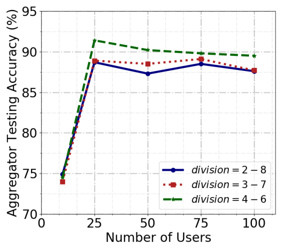
(a) Aggregator accuracy with different thresholds (MNIST).



(b) Aggregator accuracy with different thresholds (SVHN).



(c) Aggregator accuracy when data is unevenly distributed (MNIST).

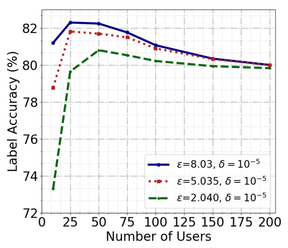


(d) Aggregator accuracy when data is unevenly distributed (SVHN).

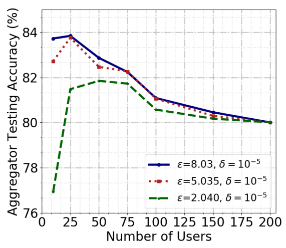
Fig. 5. (a)(b) Aggregator accuracy of MNIST and SVHN under different aggregator thresholds. (c)(d) Aggregator accuracy of MNIST and SVHN under uneven data distribution.

data are unevenly distributed. Overall for MNIST and SVHN, when data is more towards even distribution, the aggregator obtains higher accuracy. When examined more closely, we found the label accuracy almost remains the same across the three distributions, but the number of retained samples after aggregation decreases as the distribution is more towards uneven, by the results in Table III. Thus the lower accuracy of the uneven case should be attributed to a reduction of retained samples, not the label accuracies. The experimental results on CelebA, shown by Fig. 6(c) and 6(d), also support this point. It concludes that our consensus protocol effectively selects labels even when the dataset is highly unbalanced.

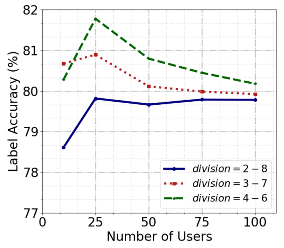
We also conduct experiments on CelebA, and the results are shown in Fig. 6(a), 6(b), 6(c) and 6(d). Interestingly, we observed when the number of users is 50, 75, 100 and the dataset is divided into $20\%/80\%$, the aggregator rapidly overfits and thus leads to a lower testing accuracy (Fig. 2(d)). As the aggregator's samples are inspected, it is found that their label vectors are highly similar with 97% likeness. The simple pattern of training samples may explain why the aggregator overfits. When further examined, the samples in CelebA are quite sparse, *i.e.,* most attributes are negative except for few positive ones (positive means the existence of the attributes). Thus when the data is unevenly distributed, the positive attributes can only be learned by a few users, and such results may be discarded since they deviates from the consensus. This may explain why the aggregator accuracies almost always decrease with the increase of the number of users on the uneven data distribution of CelebA.



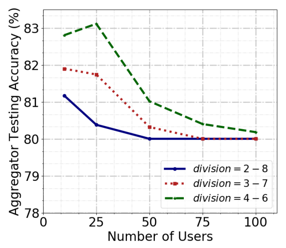
(a) Label accuracy when data is evenly distributed (CelebA).



(b) Aggregator accuracy when data is evenly distributed (CelebA).



(c) Label accuracy when data is unevenly distributed (CelebA).



(d) Aggregator accuracy when data is unevenly distributed (CelebA).

Fig. 6. Experimental results on CelebA: Aggregator accuracy decreases with the number of users when data is unevenly distributed.

VII. Conclusion

Based on Aggregation of Teacher Ensembles, we propose a privacy-preserving method to achieve consensus in the decentralized learning setting, with improved aggregator model performance and minimal information leakage. The protocol aggregates local votes in blind to label the training instances, and only when the aggregation result surpasses a threshold, meaning that consensus is achieved, can the instance be labeled. Otherwise, the instance is discarded. Our protocol ensures the only information revealed is the label with the highest noisy votes, with differential privacy guaranteed. Experimental results show that our consensus protocol achieves an improved accuracy across different datasets while preserving user privacy.

References

[1] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data," in *Proc. of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[2] ——, "Scalable Private Learning with PATE," in *Proc. of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[3] M. Pathak, S. Rane, and B. Raj, "Multiparty Differential Privacy via Aggregation of Locally Trained Classifiers," in *Advances in Neural Information Processing Systems (NIPS)*, 2010, pp. 1876–1884.

[4] A. Rajkumar and S. Agarwal, "A Differentially Private Stochastic Gradient Descent Algorithm for Multiparty Classification," in *Artificial Intelligence and Statistics*, 2012, pp. 933–941.

[5] M. Heikkilä, E. Lagerspetz, S. Kaski, K. Shimizu, S. Tarkoma, and A. Honkela, "Differentially Private Bayesian Learning on Distributed Data," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 3229–3238.

[6] R. Shokri and V. Shmatikov, "Privacy-preserving Deep Learning," in *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 1310–1321.

[7] X. Zhang, S. Ji, H. Wang, and T. Wang, "Private, Yet Practical, Multiparty Deep Learning," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1442–1452.

[8] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[9] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation," in *Theory of Cryptography Conference*. Springer, 2006, pp. 285–304.

[10] T. Nishide and K. Ohta, "Multiparty Computation for Interval, Equality, and Comparison without Bit-Decomposition Protocol," in *International Workshop on Public Key Cryptography*. Springer, 2007, pp. 343–360.

[11] N. Kiribuchi, R. Kato, T. Nishide, T. Endo, and H. Yoshiura, "Accelerating Multiparty Computation by Efficient Random Number Bitwise-Sharing Protocols," in *International Workshop on Information Security Applications*. Springer, 2011, pp. 187–202.

[12] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and Secure Comparison for On-Line Auctions," in *Australasian Conference on Information Security and Privacy*. Springer, 2007, pp. 416–430.

[13] I. Damgard, M. Geisler, and M. Kroigard, "A correction to ëfficient and secure comparison for on-line auctions.̈" *International Journal of Applied Cryptography*, vol. 2008, no. 4, pp. 323–324, 2009.

[14] M. Fischlin, "A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires," in *Cryptographers' Track at the RSA Conference*. Springer, 2001, pp. 457–471.

[15] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, B. Dan, and N. Taft, "Privacy-Preserving Ridge Regression on Hundreds of Millions of Records," *IEEE Symposium on Security & Privacy*, pp. 334–348, 2013.

[16] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our Data, Ourselves: Privacy via Distributed Noise Generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 486–503.

[17] P. Kairouz, S. Oh, and P. Viswanath, "Secure Multi-party Differential Privacy," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2008–2016.

[18] I. Mironov, "Rényi differential privacy," in *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*. IEEE, 2017, pp. 263–275.

[19] C. Dwork, A. Roth *et al.*, "The Algorithmic Foundations of Differential Privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.