Towards Fair and Decentralized Privacy-Preserving Deep Learning

Lingjuan Lyu, *Member, IEEE*, Jiangshan Yu, Karthik Nandakumar, *Senior Member, IEEE*, Yitong Li, Xingjun Ma, and Jiong Jin, *Member, IEEE*

Abstract—In current deep learning paradigms, the standalone framework tends to result in overfitting and low utility. This problem can be addressed by either a centralized framework that deploys a central server to train a global model on the joint data from all parties, or a distributed framework that leverages a parameter server to aggregate local model updates. Server-based frameworks unfortunately suffer from the single-point-of-failure problem, and the decentralized framework is born to be resistant to it. However, all the existing collaborative learning frameworks (distributed or decentralized) have overlooked an important aspect of participation: fairness. In particular, all parties can get similar models, even the ones merely making marginal contribution with low-quality data. To address this issue, we propose a decentralized privacy-preserving deep learning framework called DPPDL. It makes the first-ever investigation on the collaborative fairness in deep learning, and proposes two novel strategies to guarantee both fairness and privacy. We experimentally demonstrate that, on benchmark image datasets, fairness, privacy and accuracy in collaborative deep learning can now be effectively achieved at the same time by our proposed DPPDL. Moreover, it provides a viable solution to detect and reduce the impact of low-quality parties in the collaborative learning system.

Index Terms—Privacy-Preserving, Deep Learning, Fairness, Encryption.

1 Introduction

1.1 Background

Deep learning has become an important technology to deal with the challenging real-world problems such as image classification [1] and speech recognition [2]. Empirical evidence has demonstrated that deep learning models can benefit significantly from large-scale datasets [3]. However, large-scale datasets are not always available for a new domain, due to the significant time and effort it takes for data collection and annotation [4], [5]. Moreover, training complex deep networks on large-scale datasets could be computationally expensive and practically unachievable by a single party. Therefore, there is a high demand to perform deep learning in a collaborative manner among a group of parties. This trend is motivated by the fact that the data owned by a single party may be very homogeneous, resulting in an overfitted model that might deliver inaccurate results when applied to the unseen data, *i.e.*, poor generalizability. In addition, decomposing and parallelizing computation among different parties could help reduce the demand for resources on any single party. However, collaboration can be greatly hindered by privacy and confidentiality restrictions of local parties [6]. More importantly, lack of fairness might discourage collaboration among parties. To the best of our knowledge, all the current research works of fairness focus on the protection of some specific attributes, or aim to reduce the variance of the accuracy distribution across participants [7], [8], while none of the previous works addressed the problem of collaborative fairness and privacy at the same time. To overcome these problems, it is essential to develop a fair and privacy-preserving collaborative learning framework that respects collaborative fairness, data privacy and utility at the same time. Moreover, we adopt an honest-but-curious setting: each party is assumed to be curious in extracting the data of other parties; and yet, it is assumed to be honest in operations. This setting is reasonable as in our scenario parties are considered as organisations such as financial or biomedical institutions acting with responsibilities by laws. However, we also explore the active adversaries and seek solutions to prevent their malicious behaviors in Section 6.

1.2 The Overview of Frameworks

In general, deep learning frameworks belong to the following categories: *Standalone framework*; central server-based frameworks including *Centralized framework* and *Distributed framework*; and *Decentralized framework*. In particular, in distributed framework and decentralized framework, parties are all involved in the global or consensus model improvement process, hence we call them collaborative deep learning frameworks. A succinct comparison among different deep learning frameworks is provided in Table 1.

Standalone framework: Parties individually train standalone models on their local training data without any collaboration, as shown in Fig. 1(a). However, standalone models might fail to generalise to the unseen data.

Centralized framework: Participants pool their data into a centralized server to train a global model, as shown in

- *L. Lyu is with the Research School of Computer Science, Australian National University, ACT, Australia, 2601. E-mail:lingjuan.lyu@anu.edu.au.*
- *J. Yu is with the Faculty of Information Technology, Monash University, Clayton, Australia. E-mail:jiangshan.yu@monash.edu.*
- *K. Nandakumar is with IBM Singapore Lab, 018983. E-mail:nkarthik@sg.ibm.com.*
- *Y. Li and X. Ma are with the School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia, 3010. E-mail: yitongl4@student.unimelb.edu.au; xingjun.ma@unimelb.edu.au.*
- *J. Jin is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia. E-mail: jiongjin@swin.edu.au.*

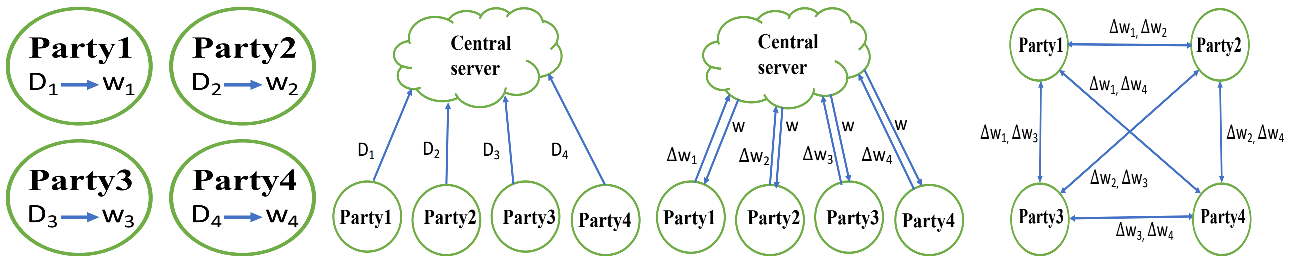

Fig. 1: (a): Standalone framework. (b): Centralized framework. (c): Distributed framework. (d): Decentralized framework (Blockchain).

Fig. 1(b). This centralized framework is very effective, but it is privacy-violating since the server is entitled to see all participants' data in the clear.

Distributed framework: Dean *et al.* [9] firstly introduced the concept of distributed deep learning, where parties collaboratively train a model by sharing local model updates with a parameter server. Distributed learning had been extensively studied in [10], [11], [12].

It should be noted that both centralized framework and distributed framework require a central server to mediate the training process, which might suffer from the following fundamental problems: (1) Party policies. Due to privacy concern, parties may not want to cede control to an untrusted server [13]; (2) Single-point-of-failure. If the central server fails or is shut down for maintenance, the whole network stops working [14].

Decentralized framework: All the above problems in the central server-based frameworks can be inherently solved by replacing the central server-based frameworks with a decentralized framework [15], which parallelizes the computation among all parties, as exemplified in Fig. 1(d). Kuo *et al.* [15] firstly proposed a decentralized machine learning framework: ModelChain, which integrates Blockchain with privacy-preserving machine learning.

In summary, all the existing collaborative frameworks (distributed or decentralized) focus on how to learn a global model or multiple local models with higher accuracy than individual standalone models, neglecting an important motivation for collaboration: **fairness**. In particular, we want to remark that there is no concrete definition of fairness in both the standalone framework and the centralized framework, because parties do not collaborate in the standalone framework, and parties cannot get access to the trained global model in the centralized framework, as the global model is only available in the form of "machine learning as a service" (MLaaS). However, in distributed framework, all parties can fetch the same global model from the central server. Similarly, in the decentralized framework, all parties can get access to the consensus model. These frameworks are obviously unfair, because in reality, some parties may contribute more high-quality data (*e.g.*, data with more diversity and high-quality annotations) than others. The reason lies in the fact that different parties may have different capacities to generate the training data, and there may exist unpredictable random errors during data collection and storage. In this work, we assume there exists certain parties with low-quality data, thus its uploaded data or model parameters in the distributed framework or the decentralized framework may disturb the learning accuracy. In the extreme case, a low-quality party might even have no data and model to start with, but still can get the same global model using current collaborative deep learning frameworks.

In this paper, we aim to reduce the impact of low-quality party on the accuracy of the learned models of other parties through quality control, which is enforced in fairness and realized through local credibility mutual evaluation mechanism, so the party with the worst performance will not dominate the whole system. In particular, we assume the low-quality parties are not malicious, *i.e.*, they follow the protocol honestly, but may attempt to infer sensitive information or benefit from other parties' data.

One naive solution to evaluate the data quality of each party is to publish all its original data or model parameters, such that the low-quality party can be easily detected, however, it has been pointed out that even model updates (gradients) can lead to a breach of privacy [16], thus not desirable.

1.3 Our Contributions

In this paper, we address all the above mentioned problems by proposing a decentralized privacy-preserving deep learning framework called DPPDL, where participants **do not** need to trust each other or any third party. The proposed DPPDL records all operations, including uploading and downloading *differentially private artificial samples* and *encrypted model updates*, as transactions. In summary, the following main contributions are made:

- We develop a decentralized privacy-preserving deep learning framework to address fairness, privacy and accuracy in collaborative deep learning at the same time.
- We are the first to formulate the notion of collaborative fairness, which is guaranteed in DPPDL through mutual evaluations of local credibility that considers the relative contribution of each party during both initial benchmarking and privacy-preserving collaborative deep learning.
- For privacy-preserving collaborative learning, instead of leveraging differential privacy at the cost of utility, we put forward a three-layer onion-style encryption scheme to guarantee both accuracy and privacy.

TABLE 1: Comparing different deep learning frameworks.

Frameworks	Standalone	Centralized [17], [18], [19]	Distributed [10], [11], [12], [20]	Decentralized [15]	Decentralized (our DPPDL)
Architecture	Fig. 1(a)	Fig. 1(b)	Fig. 1(c)	Fig. 1(d)	Fig. 1(d)
Global model	No	Yes	Yes	Depends	Depends
Local models	Yes	No	Yes	Yes	Yes
Fairness	NA	NA	No	No	Yes
Quality control	NA	No	No	No	Yes

- The experimental results on two benchmark datasets under three realistic settings demonstrate that our proposed framework achieves high fairness, delivers comparable accuracy to both centralized and distributed deep learning frameworks, and outperforms the standalone one, thus confirming the applicability of DPPDL.

2 Preliminaries

2.1 Differential Privacy

Differential privacy, as defined in Definition 1 [21], trades off privacy and accuracy by perturbing the data in a way that is (i) computationally efficient, (ii) does not allow an attacker to recover the original data, and (iii) does not severely affect the utility.

Definition 1. *A randomized mechanism $\mathcal{M}$: $\mathcal{D} \rightarrow \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (ϵ, δ)-differential privacy if for all two neighbouring inputs $D, D' \in \mathcal{D}$ that differ in one record and for any measurable subset of outputs $S \subseteq \mathcal{R}$ it holds that*

$$\Pr\{\mathcal{M}(D) \in S\} \quad \leq \quad \exp(\epsilon) \cdot \Pr\{\mathcal{M}(D') \in S\} + \delta \ .$$

Furthermore $\mathcal{M}$ is said to preserve (pure) ϵ-differential privacy if the condition holds for $\delta = 0$.

The formal definition of differential privacy has two parameters: privacy budget ϵ measures the incurred privacy leakage; δ bounds the probability that the privacy loss exceeds ϵ. The values of (ϵ, δ) accumulate as the algorithm repeatedly accesses the private data [22].

2.2 Homomorphic Encryption

Homomorphic encryption is a form of encryption that is widely used to derive the aggregate in a secure manner. Existing homomorphic encryption techniques can be categorized as fully homomorphic encryption, somewhat homomorphic encryption and partially homomorphic encryption. Fully homomorphic encryption can support arbitrary computation on ciphertexts but is less efficient [23]. On the other hand, somewhat homomorphic encryption and partially homomorphic encryption are specified by a limited number of operations [24], [25], [26], [27]. However, all these techniques generally result in longer ciphertext than the plaintext, causing extra communication cost. To counter this issue, we take inspirations from stream ciphers for efficient homomorphic-ciphertext compression [28], which also allows additive homomorphic operation over ciphertexts encrypted under different parties' keystreams, details are provided in Section 4.2.1.

2.3 Blockchain Technology

Blockchain is a decentralized (*i.e.*, a peer-to-peer, non-intermediated) system that is maintained by all the participants, called miners, in the system. The first Blockchain system is BitCoin [29], where miners create blocks by solving cryptographic puzzles through the famous proof of work mechanism. In particular, a block mainly contains the hash value of the previous block in the chain, a set of transactions organized as a hash tree, a public key of the block creator, and a random nonce. The hash value of the previous block serves as a reference to specify the state of the block, *i.e.*, it is a block created in the system right after the one being referenced. Each transaction represents a trade in the system, and the public key uniquely identifies the miner as a way for authentication. This public key is used to claim the mining reward. To spend a coin, one needs to sign the transaction by using the signing key associated with the public key that receives the coins. To create a block, a miner collects a set of valid transactions and creates its public key for this block.

Blockchain is well known for its transparency and robustness, namely, everyone is able to read and write in the Blockchain, and there is no single-point-of-failure as the Blockchain is maintained by all parties rather than a single party. Intuitively, the incremental characteristic of online deep learning makes it feasible for peer-to-peer networks like Blockchain. However, a reasonable approach to integrate Blockchain with privacy-preserving deep learning is yet to be devised.

3 DPPDL Framework

This section details our proposed Decentralized Privacy-Preserving Deep Learning (DPPDL) framework, including the mains goals of DPPDL: privacy and fairness, and an investigation on the Blockchain as the decentralized architecture for DPPDL. Tables 2 presents a number of symbols for better readability.

3.1 Main Goals of DPPDL

Privacy: In DPPDL, we assume parties do not trust each other or any third party. Hence, parties may not be willing to share their information without the promise of privacy protection. To remove the deterrents for parties to share their data, instead of publishing all the original data or model parameters, each party leverages Differentially Private GAN (DPGAN) to publish differentially private samples for mutual evaluation during initial benchmarking, and encrypts the shared gradients using three-layer onion-style encryption scheme in privacy-preserving collaborative deep learning.

TABLE 2: Table of symbols.

Symbol	Meaning
D_i, M_i	local training data and local model of party i
SD_i	μ DPGAN samples randomly chosen by party i
p_i, d_i	points and gradients download budget of party i
c_i^j, $c_i^{j\prime}$	local credibility and updated local credibility of party j given by party i
u_i	number of DPGAN samples released by party i
d_i^j	number of meaningful gradients of party j released to party i
ϵ_j	privacy level of party j
$\Delta \boldsymbol{w_j}$	gradient vector of party j
$\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}$	masked gradient vector of party j shared with party i by filling the remaining $\lvert\Delta \boldsymbol{w_j^i}\rvert - d_i^j$ gradients with 0
$\boldsymbol{w_i}$	parameter of party i at previous epoch
$\boldsymbol{w_i}'$	updated parameter of party i at current epoch
n	number of participating parties
c_{th}	lower bound of the credibility threshold
C	credible party set with local credibility above c_{th} agreed by $2/3$ parties
m_j	number of matches between majority labels and party j's predicted labels
(sk_i', pk_i')	party i's key pair for signing and verification, respectively
k_i	party i's keystream used in the first layer of three-layer onion-style encryption
fsk	fresh symmetric encryption key used in the second layer of three-layer onion-style encryption
(sk_i, pk_i)	party i's key pair for decryption and encryption in the third layer of three-layer onion-style encryption
Enc	homomorphic encryption
$Senc$	symmetric key encryption
$Aenc$	public key encryption

Fairness: The fundamental principle behind fairness is that the party who invests more time and effort to collect high-quality data should be rewarded more than the less contributive party.

Considering above goals, we design a local credibility mutual evaluation mechanism to enforce fairness in DPPDL, where parties trade their data with ”points”. The local credibility and points of each participant are initialized through an initial benchmarking, and updated through privacy-preserving collaborative deep learning. The basic idea is that participants can earn points by contributing data to the system, and use the earned points to trade data with other participants. Thus, participants are encouraged to upload more samples or gradients to earn more points (as long as it is within the limit of privacy), and make use of these points to download more gradients from others. All tradings are recorded in the immutable blockchain as transactions, providing transparency and auditability. In particular, DPPDL ensures fairness during download and upload processes as follows:

- **Download as per local credibility**: Since one party might contribute differently to different parties, the credibility of this party might be different from the view of different parties, therefore, each party i should keep a local credibility list by sorting all parties as per their local credibilities in the descending order, known only by party i itself. The higher the credibility of party j in party i's credibility list, the more likely party i will download gradients from party j, and consequently, more points will be rewarded to party j.
- **Upload as per request and privacy level**: Once one party receives download request for gradients, how many meaningful gradients will be uploaded depends on both the download request and its privacy level.

3.2 Blockchain Investigation for DPPDL

For decentralization, we investigate privacy-preserving deep learning algorithm on the private Blockchain using Blockchain 2.0, which is only available to the participating parties. Compared with the current server-based architecture, DPPDL inherits the peer-to-peer architecture of Blockchain, allowing each party to remain modular while interoperating with others. In addition, instead of ceding control to the central server, each party keeps full control of their own data, thus obeying the institutional policies. Moreover, Blockchain provides the native ability to automatically coordinate the join and departure of each party, further facilitating the independence and modularity of the participating parties. In addition, Blockchain enhances security by avoiding single-point-of-failure. Below, we design two types of blocks in the Blockchain for DPPDL, namely, *init block* and *operation block*.

An init block initializes benchmarking of the usefulness of each party's training data, as a set of init transactions. An init transaction contains the initial points that the transaction creator earns, its contributed DPGAN samples, and its public key that will be used for authenticating future transactions. The genesis block (*i.e.*, the first block) of the Blockchain is an init block, which gives the initial points and local credibilities to all the participants according to their relative contributions, as stated in Algorithm 1. If any party joins or adds new data during update, a new init block will be created and added to the existing Blockchain.

An operation block contains a set of transactions defining the UPLOAD operation and/or DOWNLOAD operation. All UPLOAD and DOWNLOAD transactions are signed by their creator using the private key associated with the public key recorded in the init transaction. An UPLOAD operation commits that a data owner has uploaded local model gradients to the party who sent a download request. A DOWNLOAD operation states that a participant is committed an order to request some local model updates from other participants. Upon receiving a DOWNLOAD transaction, Blockchain miners verify its signature, check if the requester has enough balance to download the number of requested gradients, and record successfully the verified transactions in an operation block. Once the DOWNLOAD transaction is recorded in the Blockchain, the requested local model gradients will be encrypted and uploaded by the owner to a public accessible storage, and re-encrypted using the recipient's public key defined in the DOWNLOAD transaction.

In particular, the privacy of local model gradients is protected through a three-layer onion-style encryption scheme (see Section 4.2). The first layer encrypts the local model gradients through our symmetric key based homomorphic encryption (Algorithm 3), which allows each party to learn the aggregate of the received gradients without revealing individual gradients, *i.e.*, party obliviousness. The second and third layer present a standard hybrid encryption process: the second layer uses a freshly generated symmetric key fsk to re-encrypt the first layer ciphertext, and the third layer encrypts fsk with the requester party i's public key pk_i. In this way, we minimize the required computational cost incurred by the asymmetric key based encryption. The commitment of the uploaded encrypted local model gradients (*e.g.*, hash value of the ciphertext, as presented in Fig 3) will be included in the UPLOAD transaction.

In our private Blockchain, only the requester who pays could read the plaintext; others can verify that this transaction has happened, but cannot read. When a requester dishonestly blames a data uploader, the data uploader reveals the plaintext as an evidence. In this case, the requester will be forced to pay a fine that it deposits when filing a dishonest claim. Once an UPLOAD transaction is recorded in the Blockchain, the points will be automatically transferred from the requester to the uploader. An example of INITIALIZE and DOWNLOAD transaction stored by Blockchain are shown in Fig. 2 and Fig. 3, respectively. This Blockchain is either maintained in a permissionless and distributed manner, such as by utilizing Ethereum, or in a permissioned and decentralized manner, such as IBM's Hyperledger Fabric.

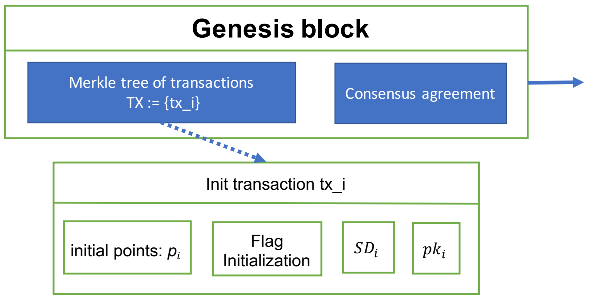

Fig. 2: **An example structure of the genesis block**. It mainly contains two key components, one is a set of init transactions organized as leaves of a Merkle tree; and the other one is the consensus agreement reached by the participants through the underlying consensus protocol (*e.g.*, PBFT or PoS), which is specific to the deployed Blockchain. The pk_i' in the init transaction is a signature verification key of party i.

4 DPPDL Realization

This section details the two-stage realization in DPPDL, including initial benchmarking and privacy-preserving collaborative deep learning, as shown in Fig. 4, followed by the quantification of fairness.

4.1 Initial Benchmarking

Initial benchmarking algorithm aims to benchmark the quality of local training data of each participant via mutual evaluation before collaborative learning starts. Otherwise, we cannot prevent a parasitic participant from gaining access to the best individual model right at the initialization. For this benchmarking, our proposed solution is as follows: each participant can train a DPGAN based on its local training data to generate artificial samples, which will be distributed and used for secondary analysis. However, these generated samples **should not** disclose the true sensitive examples, as well as the true distribution of data, but only a few implicit density estimation within a modest privacy budget used in DPGAN. Each participant publishes individually generated artificial samples as per individual privacy level without releasing labels. All the other participants produce predictions for the received artificial samples using their pre-trained standalone models and send the predicted labels back to the party who generated these samples. The aim of publishing artificial samples generated by DPGAN is two-fold:

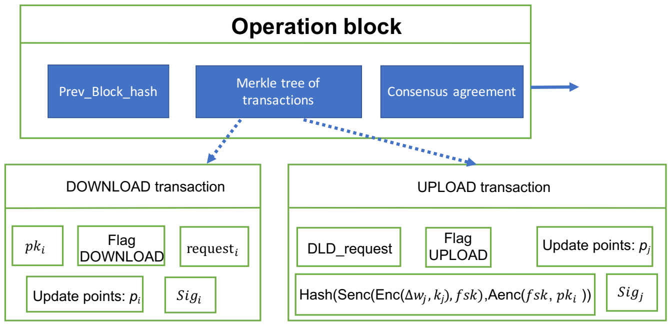

Fig. 3: **An example structure of the operation block**. It mainly contains three key components, namely, the hash value Prev_Block_hash of the previous block, a set of UPLOAD/DOWNLOAD transactions organized as a Merkle tree, and the consensus agreement of this block. In particular, a Prev_Block_hash links the current block to the previous one, and the request in the UPLOAD transaction acts as a reference to the associated DOWNLOAD transaction. pk_i in the DOWNLOAD transaction is the public key that will be used in the last layer of our three-layer onion-style encryption scheme, $request_i$ is a unique request ID of this transaction and will be referenced in the corresponding UPLOAD transaction via DLD_request, and Sig_i is the signature on this transaction. Enc, $Senc$, and $Aenc$ refer to homomorphic encryption, symmetric key encryption, and public key encryption, respectively.

- **To get prior information about individual models before collaborative learning starts**. If a participant does not have reasonable amount of training data to produce a decent model, it will perform poorly in the initial evaluation of DPGAN samples and be ranked low by other participants, therefore, other participants will be cautious in sharing gradients with such a participant.
- **To get a rough estimate of data distribution of other participants**. Two participants can mutually benefit only if their data distributions are different but have some degree of overlap. Suppose that two participants A and B have published almost identical artificial samples, it means that their training data distribution are almost identical. In this case, the updates from B are unlikely to increase the accuracy of model A and vice versa. Consequently, their models are unlikely to improve significantly by incorporating updates from each other. Therefore, during the

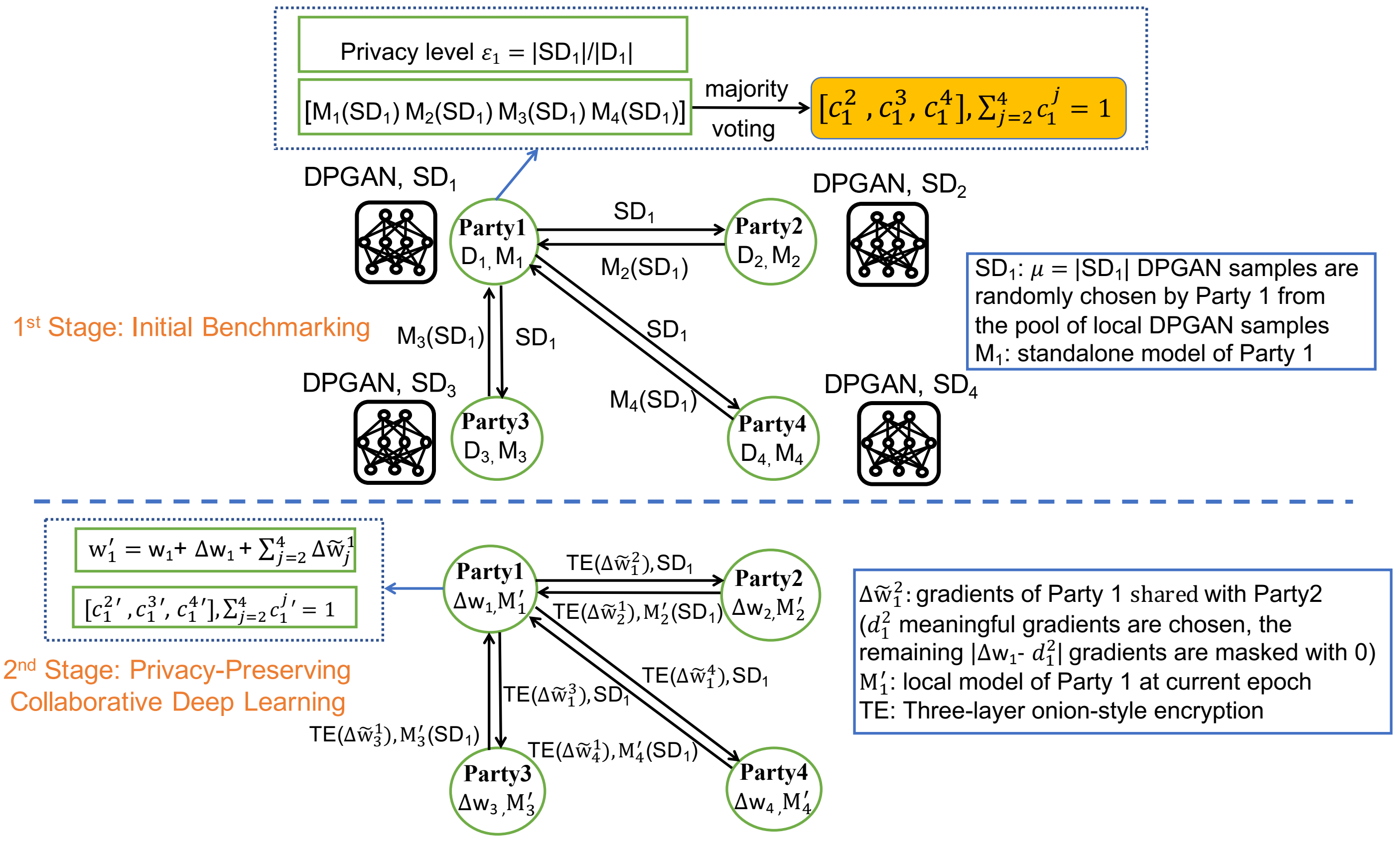
Privacy level $\varepsilon_1 = |SD_1|/|D_1|$
$[M_1(SD_1)\ M_2(SD_1)\ M_3(SD_1)\ M_4(SD_1)]$
majority voting
$[c_1^2, c_1^3, c_1^4], \sum_{j=2}^4 c_1^j = 1$
DPGAN, SD_1
DPGAN, SD_2
DPGAN, SD_3
DPGAN, SD_4
Party1 D_1, M_1
Party2 D_2, M_2
Party3 D_3, M_3
Party4 D_4, M_4
SD_1
$M_2(SD_1)$
$M_3(SD_1)$
$M_4(SD_1)$
1st Stage: Initial Benchmarking
SD_1: μ = $|SD_1|$ DPGAN samples are randomly chosen by Party 1 from the pool of local DPGAN samples
M_1: standalone model of Party 1
$w_1' = w_1 + \Delta w_1 + \sum_{j=2}^4 \Delta\tilde{w}_j^1$
$[c_1^{2\prime}, c_1^{3\prime}, c_1^{4\prime}], \sum_{j=2}^4 c_1^{j\prime} = 1$
2nd Stage: Privacy-Preserving Collaborative Deep Learning
Party1 $\Delta w_1, M_1'$
Party2 $\Delta w_2, M_2'$
Party3 $\Delta w_3, M_3'$
Party4 $\Delta w_4, M_4'$
$TE(\Delta\tilde{w}_1^2), SD_1$
$TE(\Delta\tilde{w}_2^1), M_2'(SD_1)$
$TE(\Delta\tilde{w}_1^3), SD_1$
$TE(\Delta\tilde{w}_3^1), M_3'(SD_1)$
$TE(\Delta\tilde{w}_1^4), SD_1$
$TE(\Delta\tilde{w}_4^1), M_4'(SD_1)$
$\Delta\tilde{w}_1^2$: gradients of Party 1 shared with Party2 (d_1^2 meaningful gradients are chosen, the remaining $|\Delta w_1 - d_1^2|$ gradients are masked with 0)
M_1': local model of Party 1 at current epoch
TE: Three-layer onion-style encryption

Fig. 4: Two-stage Realization in DPPDL.

subsequent epochs, A and B should mutually avoid downloading updates from each other. Other participants can choose to download updates from either A or B but not both. On the other hand, suppose that two participants A and B have completely different data distributions, similarly, the updates from B are unlikely to increase the accuracy of model A and vice versa. Thus during the subsequent epochs, A and B should also try to avoid downloading updates from each other. Furthermore, suppose that A's data distribution is different from that of all the other participants, all these participants should try to avoid A. This automatically takes care of the scenario where a honest participant publishes some gradients, while all the other honest participants assign very low credibility to the publisher. In this case, the data distribution of the publisher is completely different from that of the other participants, hence it is still reasonable to reduce the credibility of the publisher, because other participants are anyway unlikely to gain much from its published gradients.

We next describe the detailed procedures of initial benchmarking in Algorithm 1, including: local credibility initialization, and privacy level and points initialization.

4.1.1 *Local Credibility Initialization*

For local credibility initialization, each party compares the majority voting of all the combined labels with a particular party's predicted labels to evaluate the effect of this party. It relies on the fact that the majority voting of all the combined labels reflects the outcome of the majority of parties, while the predicted labels of party j only reflects the outcome of party j. For example, in the case of party i initializing local credibility list for other parties, party i broadcasts its DPGAN samples to other parties, who label these samples using their pre-trained standalone models, and send the corresponding predicted labels back to party i. Meanwhile, party i also labels its own artificial samples using its pre-trained standalone model, then combines all parties' predicted labels as a label matrix with total n columns, where each column corresponds to one party's predicted labels. Party i then initializes the local credibility of party j as follows:

$$c_i^j = \frac{m_j}{u_i} \tag{1}$$

where m_j is the number of matches between the majority labels and party j's predicted labels, and u_i is the number of DPGAN samples released by party i. Afterwards, party i normalizes c_i^j within [0,1]. If the majority of parties report that the local credibility of one participant is lower than the threshold c_{th}, implying a potentially low-quality contributor, it will be banned from the local credibility lists of all parties. Here, c_{th} should be agreed by the majority of parties. In the update process, party i is more likely to download gradients from more credible participants, while download less, even ignore those published by less credible participants.

Algorithm 1 Initial Benchmarking

```
Input: number of participating parties $n$, C={1,...,n}
Output: local credibility and points of all parties
1: Pre-train aprior models: Each party $i$ trains standalone model $M_i$ and local DPGAN based on its local training data.
2: Privacy level initialization: During initialization, party $i$ randomly selects and releases $u_i$ artificial samples generated by local DPGAN to any party $j$, privacy level is autonomously determined as $\epsilon_i = u_i/|D_i|$, where $|D_i|$ is local training data size of party $i$.
3: Local credibility initialization: Party $j$ labels the received artificial samples by its local model $M_j$, then returns the predicted labels back to party $i$. Meanwhile, party $i$ also labels its own DPGAN samples using $M_i$. Afterwards, party $i$ applies majority voting to all the predicted labels, then initializes the local credibility of party $j$ as $c_i^j = \frac{m_j}{u_i}$, where $m_j$ is the number of matches between majority labels and party $j$'s predicted labels, and $u_i$ is the number of DPGAN samples released by party $i$. The detailed explanation is elaborated in Section 4.1.1.
4: Local credibility normalization: $c_i^j = \frac{c_i^j}{\sum_{j \in C \setminus i} c_i^j}$
if $c_i^j < c_{th}$ then
    party $i$ reports party $j$ as low-quality contributor
end if
5: Credible party set: If the majority of parties report party $j$ as low-quality, Blockchain removes party $j$ from the credible party set $C$ and all parties run step 4 again.
6: Points initialization to download gradients: $p_i = \epsilon_i * |\boldsymbol{w_i}| * (n-1)$.
```

4.1.2 *Privacy Level and Points Initialization*

Based on the number of artificial samples u_i that party i publishes at the beginning, privacy level of party i is autonomously determined that it is comfortable with, which can be quantified as $\epsilon_i = u_i/|D_i|$, where D_i is the local training data of party i. The more private party tends to release less samples, while the less private party is comfortable with releasing more samples. Similarly, during collaborative learning process, more private party would prefer to release less gradients. Point is initialized as follows:

$$p_i = \epsilon_i * |\boldsymbol{w_i}| * (n-1) \tag{2}$$

where ϵ_i is the privacy level of party i, $|\boldsymbol{w_i}|$ is the number of model parameters, and n is the number of parties. The gained points from initial benchmarking will be used to download gradients in the follow-up collaborative learning process, and how many gradients will be downloaded is dependent on both the local credibility and privacy level of the requested party.

Differentially Private GAN (DPGAN): During initial benchmarking, although each party only releases a small amount of unlabeled samples as per individual privacy level, it may still implicitly disclose privacy of local training data. The practice of generating samples under differential privacy with generative adversarial network (GAN) offers a technical solution for those who wish to share data to the challenge of privacy. Therefore, we are inspired to train a *Differentially Private GAN* (DPGAN) by adding tailored noise to the gradients during DPGAN learning [30] at each party. In the context of a GAN, the discriminator is the only component that accesses the private real data. Therefore, we only need to train the discriminator under differential privacy. The differential privacy guarantee of the entire GAN directly follows because the computations of the generator are simply post-processing from the discriminator. The main idea follows the post-processing property of differential privacy [21], as stated in Lemma 1. **To counter the stability and scalability issues of training DPGAN models**, we apply multi-fold optimization strategies, including weight clustering, adaptive clipping, and warm starting, which significantly improve both training stability and utility, details can be referred to [30]. Unlike PATE framework in [31], where privacy loss is proportional to the amount of data needed to be labeled in public test data, differentially private generator can generate infinite number of samples for the intended analysis, while rigorously guaranteeing (ϵ, δ)-differential privacy of training data. Without loss of generality, we exemplify DPGAN in the context of the improved WGAN framework [32] and let each party generate total 1000 artificial samples. As demonstrated by Zhang *et al.* [30], DPGAN is able to synthesize both grey image and RGB image with inception scores fairly close to the real data and samples generated by regular GANs without any privacy protection.

Lemma 1. *Let algorithm $\mathcal{A} : \mathbb{R}^n \rightarrow \mathbb{R}$ be a randomized algorithm that is (ϵ, δ)-differentially private. Let $f : \mathbb{R} \rightarrow \mathbb{R}'$ be an arbitrary randomized mapping. Then*

$$f \circ \mathcal{A} : \mathbb{R}^n \rightarrow \mathbb{R}'$$

is (ϵ, δ)-differentially private.

Meanwhile, it is well-known that larger amount of training data causes less privacy loss, and allows for more iterations within a moderate privacy budget [22]. Due to the scarcity of training data of each party, **data augmentation** is exploited to expand local data size of each party to 100 times, which allows DPGAN to generate realistic samples within a moderate privacy budget. In particular, we augment original data with rotation range of 1 and width shift range and height shift range of 0.01. In our experiment, we use moments accountant described in [22] to track the spent privacy over the course of training. Our DPGAN is able to generate realistic MNIST samples with $\epsilon = 4$ and $\delta = 10^{-5}$, as shown in Fig. 5. Note that each party can individually train DPGAN and generate massive DPGAN samples offline without affecting collaboration.

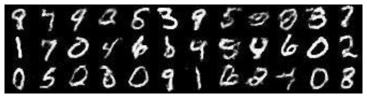

Fig. 5: Generated DPGAN samples with $\epsilon = 4, \delta = 10^{-5}$ using the augmented 60000 MNIST examples of one party who owns 600 original MNIST examples.

4.2 Privacy-Preserving Collaborative Deep Learning

Algorithm 2 summarizes steps for the privacy-preserving collaborative deep learning, including how to update points as per upload/download, how to preserve privacy of individual model updates using three-layer onion-style encryption, followed by parameter and local credibility update,

Algorithm 2 Privacy-Preserving Collaborative Deep Learning

```
Input: $C$, $c_i^j$, $p_i$, $p_j$, $d_i$, $\epsilon_j$, $\boldsymbol{w_i}$, $\Delta\boldsymbol{w_j}$
Output: updated points $p_j'$, $p_i'$, parameters $\boldsymbol{w_i}'$, and local credibility c˙i˙j′
1: Points update: In each epoch, party $i$ aims to download total $d_i$ gradients from all parties in $C$, while party $j \in C$ can at most provide $\epsilon_j \times |\Delta\boldsymbol{w_j}|$ gradients, one point is consumed/rewarded for each download and upload. Each party $i$ updates local model parameters based on the gradients of party $j \in C$ as follows:
if $d_i < p_i$ then
    for $j \in C$ do
        $d_i^j = min(c_i^j * d_i, \epsilon_j * |\Delta\boldsymbol{w_j}|)$
        $p_j' = p_j + d_i^j$, $p_i' = p_i - d_i^j$
        $\Delta\boldsymbol{w_j^i} = \Delta\boldsymbol{w_j}$, party $j$ first chooses $d_i^j$ meaningful gradients from $\Delta\boldsymbol{w_j^i}$ according to "largest values" criterion: sort gradients in $\Delta\boldsymbol{w_j^i}$ and choose $d_i^j$ of them, starting from the largest, then masks the remaining $|\Delta\boldsymbol{w_j^i}| - d_i^j$ gradients with 0 as $\Delta\tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}$.
    end for
end if
2: Three-layer onion-style encryption: Party $j$ follows Algorithm 3 to encrypt the masked gradients $\Delta\tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}$ with its keystream $k_j$ as $c = Enc(\Delta\tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}, k_j)$, and re-encrypts the encrypted gradients $c$ with a fresh symmetric encryption key $fsk$ as $Senc(c, fsk)$, the symmetric encryption key of the second layer is encrypted in the third layer by the receiver party $i$'s public key $pk_i$ as $Aenc(fsk, pk_i)$. Finally, the two-layer encrypted gradients $Senc(c, fsk)$ and the encrypted fresh symmetric encryption key $Aenc(fsk, pk_i)$ are sent to party $i$;
3: Parameter update: party $i$ uses the paired secret key $sk_i$ to decrypt the received encrypted fresh symmetric encryption key as $fsk$, then uses $fsk$ to decrypt the two-layer encrypted gradients as $c = Enc(\Delta\tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}, k_j)$, finally decrypts the sum of all the received gradients using homomorphic property and updates local parameters by integrating all its plain gradients $\Delta\boldsymbol{w_i}$ as: $\boldsymbol{w_i}' = \boldsymbol{w_i} + \Delta\boldsymbol{w_i} + Dec(\sum_{j \in C \setminus i} Enc(\Delta\tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}, k_j), -k_i) = \boldsymbol{w_i} + \Delta\boldsymbol{w_i} + \sum_{j \in C \setminus i} \Delta\tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}$, where $\boldsymbol{w_i}$ is party $i$'s local parameters at previous epoch.
4: Local credibility update: party $i$ randomly selects and releases $u_i$ artificial samples to any party $j$ for labelling, mutual evaluation is repeated by following Step 3 of Algorithm 1 to calculate local credibility of party $j$ at current epoch as $c_i^{j\prime}$. Party $i$ updates local credibility of party $j$ by integrating its historical credibility as:
        $$c_i^{j\prime} = 0.2 * c_i^j + 0.8 * c_i^{j\prime}$$
   where $c_i^j$ is the local credibility of party $j$ at previous epoch.
5: Local credibility normalization: $c_i^{j\prime} = \frac{c_i^{j\prime}}{\sum_{j \in C} c_i^{j\prime}}$
if $c_i^{j\prime} < c_{th}$ then
    party $i$ reports party $j$ as low-quality contributor
end if
6: Credible party set: If the majority of parties report party $j$ as low-quality, Blockchain removes party $j$ from credible party set $C$ and all parties run Step 5 again.
```

and credible party set maintenance. We discuss the most important details for parameter update, three-layer onion-style encryption, and local credibility update as follows.

4.2.1 Parameter Update with Homomorphic Encryption

Sharing gradients can prevent direct exposure of the local data, but may indirectly disclose the sensitive information of local data. To further avoid potential privacy leakage from sharing gradients and facilitate gradients aggregation during the collaborative learning process, we use additive homomorphic encryption, such that each party can only decrypt the sum of all the received encrypted gradients, but cannot access any of them. Specifically, Vernam cipher or one-time pad (OTP) has been mathematically proved to be completely secure, which cannot be broken given enough ciphertext and time. Therefore, we use simple and provably secure OTP for additively homomorphic encryption that allows efficient aggregation of encrypted data [33], [34]. The main idea of forming the ciphertext is to combine the keystream with the plaintext digits. Meanwhile, rather than XOR operation typically found in stream ciphers, which is unsecured under the frequency analysis attacks, our encryption scheme uses modular addition (+), and is hence very efficient [33]. The security relies on two important features: (1) the keystream changes from one message to another; and (2) all the operations are performed modulo a large integer M [33].

The detailed procedure for homomorphic encryption is presented in Algorithm 3. In practice, if p = max(x_i), M is derived as $M = 2^{\lceil log_2(p \times n) \rceil}$. All computations in the remainder of this paper are modulo M unless otherwise noted. However, all the original floating-point values need to be mapped to the discrete domain of integers using Scaling, Rounding, Unscaling (SRU) algorithm [34]. A pseudorandom keystream k can be generated by a secure pseudo random function (PRF) by implementing a secure stream cipher, such as Trivium [35], keyed with each party's keystream k_i and a unique message ID. For encryption purpose, the secret keys are pre-computed through a trusted setup, which can be performed by a trusted dealer or through a standard SMC protocol. For example, a trusted key managing authority can generate these keystreams in each epoch of information exchange, but the generated keystreams cannot be used more than once. The trusted setup generates non-zero random shares of 0: $\sum_{i \in C} k_i = 0$, such that each participant $i \in C$ obtains a keystream k_i. It should be noted that if Blockchain removes party j from the credible party set C, the trusted setup should be restarted among the remaining parties to construct the new credible party set C.

Algorithm 3 Homomorphic Encryption Scheme

```
Setup
1: A trusted dealer randomly generates $|C|$ keystreams: $k_1, \ldots, k_{|C|} \in [0, M-1]$, such that $\sum_{i \in C} k_i$ (mod $M$)= 0, where $M$ is a large integer.
2: Party $i$ obtains keystream $k_i$.

Enc($m$, $k$)
1: Represent message $m$ as integer $m \in [0, M-1]$.
2: Let $k$ be a randomly generated keystream, where $k \in [0, M-1]$.
3: Compute $c = Enc(m, k) = m + k$.

Dec($c$, $k$)
1: $Dec(c, k) = c - k$.

AggrDec($k_i$)
1: Let $c_j = Enc(m_j, k_j)$, where $j \in C \setminus i$.
2: Party $i$ uses $-k_i = \sum_{j \in C \setminus i} k_j$ to decrypt the aggregation of other parties as follows: $Dec(\sum_{j \in C \setminus i} c_j, -k_i) = \sum_{j \in C \setminus i} c_j - \sum_{j \in C \setminus i} k_j = \sum_{j \in C \setminus i} m_j$.
```

Model parameter of party i is updated as per gradients-encrypted SGD as follows:

$$\begin{aligned} \boldsymbol{w_i}' &= \boldsymbol{w_i} + \Delta \boldsymbol{w_i} + Dec(\textstyle\sum_{j \in C \setminus i} Enc(\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}, k_j), -k_i) \\ &= \boldsymbol{w_i} + \Delta \boldsymbol{w_i} + \textstyle\sum_{j \in C \setminus i} \Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}} \end{aligned}$$

where Enc and Dec correspond to encryption and decryption operations in Algorithm 3, $\boldsymbol{w_i}$ is the local parameters of party i at previous epoch, $\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}$ is the masked gradient vector of party j shared with party i, where only d_i^j gradients are meaningful, *i.e.*, d_i^j elements of total $|\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}|$ elements are kept intact, while the remaining $|\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}| - d_i^j$ elements are nullified as 0. The second equality follows the homomorphic addition property, thus participant i can get the updated $\boldsymbol{w_i}'$ correctly after decryption, without having access to either $\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}^{\boldsymbol{i}}$ or $\Delta \boldsymbol{w_j}$. DPPDL ensures party obliviousness by capturing the following security notions:

- Each participant knows nothing but the sum of its received gradients in each round of communication, and cannot infer any information about other participants' data.
- If several participants form a coalition against the remaining participants, or if a subset of the encrypted data has been leaked, then each participant can inevitably learn the sum of gradients received from the remaining participants. In this case, each participant learns no additional information about the remaining participants' data.

4.2.2 Three-layer Onion-style Encryption

However, as all parties need to store different encrypted gradients that are meant to be sent to different parties on Blockchain for commitment, all the encrypted gradients are also accessible to all parties. Applying public-key encryption on top of homomorphic encryption for authentication [34] could counter this problem, however, as the released gradient vector is high-dimensional, encrypting gradient vector is both computation and communication expensive. Therefore, we propose a three-layer onion-style encryption scheme. In more details, the first layer protects local model gradients by using symmetric key keystream k_j for homomorphic encryption, as presented in Algorithm 3. The second layer and the third layer are classic hybrid encryption, as used in OpenPGP [36] for instance. In particular, in the second layer, a fresh symmetric encryption key fsk will be generated and used to re-encrypt the ciphertext of the first layer, and then the fresh symmetric key is encrypted by using the receiver's public key pk_i in the third layer. In this way, the encryption of large-scale data becomes very effective, and the receiver could be authenticated as well: only the receiver who has the corresponding secret key sk_i paired with the public key pk_i can decrypt the two-layer encrypted gradients committed on the Blockchain.

4.2.3 Local Credibility Update

Instead of using the standalone models as in the local credibility initialization, during each epoch of collaborative learning, each party randomly selects and shares a subset of DPGAN samples as per individual privacy level, then calculates the local credibility of other parties based on the returned labels, which are evaluated by using its updated local model at current epoch. The mutual evaluation follows the similar procedure as in Step 3 of Algorithm 1. Finally, local credibility of each party is updated by integrating its historical local credibility as per Step 4 of Algorithm 2. In this way, local credibility of each party can be correspondingly updated, reflecting more accurately how much one party contributes to different parties in collaborative learning.

4.3 Quantification of Fairness

Fairness of our framework can be quantified by the correlation coefficient between individual contribution (X axis) and model test accuracy (Y axis). The X axis represents the contribution of each party, characterized by their privacy levels or the sizes of training data, as the party who is less private or has more data empirically contributes more. Y axis refers to the final test accuracy, which measures the performance of individual model after collaboration, and is expected to be positively correlated with X axis to deliver good fairness. Conversely, negative coefficient implies bad fairness. Equation 3 formally quantifies fairness:

$$r_{xy} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{(n-1) s_x s_y} \tag{3}$$

where $\bar{x}$ and $\bar{y}$ are the sample means of $\boldsymbol{x}$ (privacy levels or the sizes of training data of different parties) and $\boldsymbol{y}$ (test accuracies of individual models), s_x and s_y are the corrected standard deviations. The range of fairness is within [-1,1], with higher values implying better fairness.

5 Performance Evaluation

5.1 Datasets

We implement experiments on two benchmark image datasets. The first is the MNIST dataset [37] for handwritten digit recognition consisting of 60,000 training examples and 10,000 test examples. Each example is a 32x32 gray-level image [11], with digits locating at the center of the image. The second is the SVHN dataset [38] of house numbers obtained from Google's street view images, which contains 600,000 training examples, from which we use 100,000 for training and 10,000 for testing. Each example is a 32x32 centered image with three channels (RGB). SVHN is more challenging as most of the images are noisy, and contain distractors at the sides. The size of the input layer of neural networks for MNIST and SVHN are 1024 and 3072, respectively. The objective is to classify the input as one of 10 possible digits within ["0"-"9"], thus the size of the output layer is 10. We normalize the training examples by subtracting the average and dividing by the standard deviation of training examples.

5.2 SGD Frameworks

We demonstrate the effectiveness of our proposed DPPDL by comparison with the following three frameworks. In all frameworks, SGD is applied to each party.

Standalone framework assumes parties train standalone models on local training data without any collaboration. This framework delivers maximum privacy, but minimum utility, because each party is susceptible to falling into local optima when training alone.

Centralized framework allows a trusted server to have access to all participants' data in the clear, and train a global model on the combined data using standard SGD. Hence, it is a privacy-violating framework.

Distributed framework enables parties to train independently and concurrently, and chooses a fraction of parameters to be uploaded at each iteration. In particular, as shown in [11], Distributed Selective SGD (DSSGD) can achieve even higher accuracy than the centralized SGD because updating only a small fraction of parameters at each epoch acts as a regularization technique to avoid overfitting by preventing the neural network weights from jointly "remembering" the training data. Hence, we take DSSGD for the analysis of the distributed framework. As DSSGD with round robin parameter exchange protocol results in the highest accuracy [11], we follow the round robin protocol for DSSGD, where participants run SSGD sequentially, each downloads a fraction of the most updated parameters from the server, runs local training, and uploads selected gradients; the next party follows in the fixed order. Gradients are uploaded according to the "largest values" criterion. It should be noted that in round robin protocol, the last party who downloads parameters always gains much more information than previous parties, which is obviously unfair.

5.3 Experimental Setup

For local model architecture, we consider two popular neural network architectures: *multi-layer perceptron* (MLP) and *convolutional neural network* (CNN), **these two model architectures are the same as in [11]**. For local model training, we set the learning rate as 0.001, learning rate decay as 1e-7, and batch size as 1. In addition, to reduce the impact of different initializations and avoid non-convergence, each party is initialized with the same parameter w_0, then local training is run on individual training data to update local model parameter $\boldsymbol{w_i}$. To speed up convergence, we let each party individually train 10 epochs before collaborative learning starts. For all the experiments, we set the local credibility threshold as $c_{th} = \frac{1}{n} * \frac{2}{3}$, where n is the number of parties. Next, we investigate three realistic IID settings as follows:

Same privacy level: in the first case, privacy level of each party is set as 0.1, *i.e.*, each party only releases 10% meaningful gradients during collaboration. For each party, we randomly sample 1% of the entire database as local training data, *i.e.*, 600 examples for MNIST and 1000 examples for SVHN;

Different privacy level: in the second case, privacy level of each party is randomly sampled from $[0.1, 0.5]$, and parties release meaningful gradients as per individual privacy level during collaboration. For each participant, we randomly sample 1% of the entire database as local training data as above.

Imbalanced partition: in the third case, we simulate the case where different parties have different data size. In particular, for MNIST dataset, we randomly partition total $\{$2400, 9000, 18000, 30000$\}$ examples among $\{$4,15,30,50$\}$ parties respectively. Similarly, for SVHN dataset, total $\{$4000, 15000, 30000, 50000$\}$ examples are randomly partitioned among $\{$4,15,30,50$\}$ parties respectively. The privacy level of each party is fixed to 0.1.

5.4 Experimental Results

TABLE 3: Fairness of distributed framework and our DPPDL over MNIST dataset, with different model architectures, different party numbers (P-k) and different settings as described in Section 5.3.

	Different privacy level				Imbalanced partition			
	Distributed		DPPDL		Distributed		DPPDL	
	CNN	MLP	CNN	MLP	CNN	MLP	CNN	MLP
P4	-0.68	0.30	**0.89**	**0.84**	-0.97	0.28	**0.92**	**0.96**
P15	0.20	-0.15	**0.76**	**0.79**	0.03	-0.07	**0.90**	**0.83**
P30	-0.02	0.02	**0.79**	**0.84**	0.13	0.01	**0.75**	**0.63**
P50	-0.16	-0.05	**0.75**	**0.67**	0.14	0.07	**0.72**	**0.57**

TABLE 4: Fairness of distributed framework and our DPPDL over SVHN dataset, with different model architectures, different party numbers (P-k) and different settings as described in Section 5.3.

	Different privacy level				Imbalanced partition			
	Distributed		DPPDL		Distributed		DPPDL	
	CNN	MLP	CNN	MLP	CNN	MLP	CNN	MLP
P4	0.27	0.26	**0.75**	**0.73**	0.38	0.20	**0.92**	**0.91**
P15	0.16	0.19	**0.74**	**0.71**	-0.13	0.36	**0.80**	**0.88**
P30	-0.14	0.12	**0.58**	**0.65**	0.04	-0.27	**0.61**	**0.78**
P50	-0.25	-0.37	**0.67**	**0.66**	-0.23	0.15	**0.68**	**0.69**

For collaborative fairness comparison, we only analyze our DPPDL and the distributed framework using DSSGD, neglecting centralized framework and standalone framework, because parties cannot get access to the trained global model in the centralized framework, while parties do not collaborate in the standalone framework. Table 3 and Table 4 list the calculated fairness of the distributed framework and our DPPDL over MNIST and SVHN datasets, with different architectures, different party numbers and different settings, as detailed in Section 5.3. In particular, we omit the results for the same privacy level setting, as both distributed framework and our DPPDL can achieve similarly well fairness. All the results for the setting of different privacy level and imbalanced partition are averaged over five trails. As is evidenced by the high positive values of fairness, with most of them above 0.5, DPPDL achieves reasonably good fairness, confirming the intuition behind fairness: the party who is less private and has more training data delivers higher accuracy. In contrast, the distributed framework exhibits bad fairness with significantly lower values than that of DPPDL in all cases, and even negative values in some cases, manifesting the lack of fairness in the distributed framework. This is because in the distributed framework, all the participating parties can derive similarly well models, no matter how much one party contributes.

For accuracy comparison, following [11], we report the best accuracy when running the distributed framework using DSSGD and our DPPDL on MNIST dataset. In particular, we implement DPPDL using synchronous SGD protocol under the setting of same privacy level in Section 5.3, *i.e.*, each party is associated with 600 examples, and the privacy level of each party is set as 0.1 ($\epsilon_j = 0.1$). For the distributed framework, we use DSSGD by following round robin protocol, upload rate is set as 0.1 ($\theta_u = 0.1$). Fig. 6 presents the best accuracy we obtain when running different frameworks with MLP architecture. It can be observed that DPPDL does not sacrifice model utility by a considerable margin compared with the distributed framework using DSSGD without differential privacy and the centralized framework, and consistently delivers better accuracy than the standalone SGD. Beyond 100 epochs, we can potentially achieve slightly higher accuracy.

Table 5 provides the accuracy on MNIST dataset of {4,15,30,50} parties using three baseline frameworks, and our proposed DPPDL in three realistic settings in Section 5.3. Here we report the best accuracy because fairness enables each party to get a different local model after collaborative learning, and we expect the most contributive party derives a local model with maximum accuracy approximating the non-private centralized and distributed frameworks. Similarly, Table 6 provides the accuracy on SVHN dataset. For both MNIST and SVHN datasets using CNN and MLP architectures, we show the worst accuracy for standalone SGD (minimum utility, maximum privacy). In particular, DPPDL obtains comparable accuracy to both the centralized framework and the distributed framework using DSSGD without differential privacy, and always achieves higher accuracy than the standalone SGD. For example, as shown in Table 5, for MNIST dataset of 50 parties with CNN model, our DPPDL achieves 98.07%-98.22% test accuracy under different settings, which is higher than the standalone SGD 94.05%, and comparable to 98.83% of the distributed framework using DSSGD without differential privacy, and 98.58% of the centralized framework.

The above fairness results in Table 3 and Table 4, and accuracy results in Table 5 and Table 6 demonstrate that **our proposed framework DPPDL achieves reasonable fairness, at the expense of a tiny decrease in model utility**.

Complexity Analysis. The main communication cost occurs when each party sends its encrypted gradients to the other $(n-1)$ parties, resulting in $(n-1)*L$ ciphertexts, where n and L are the number of parties and the size of the released gradients (the encrypted symmetric key size is negligible compared with the encrypted gradients). Therefore, our framework is applicable to practical applications such as biomedical or financial institutions where the number of parties is not too huge. On the other hand, the main computation cost occurs at each party who needs to train a local DPGAN during initial benchmarking, compute local gradients, and conduct three-layer onion-style encryption during collaborative deep learning. However, all parties can individually train their DPGAN models offline before collaborative deep learning starts, and all parties can individually train local models in parallel, hence deep learning computation cost is not an obstacle for those parties with enough computational power. Moreover, our encryption scheme using stream ciphers and hybrid encryption is relatively efficient, because encrypting a short plaintext (*i.e.*, the symmetric key) requires only one asymmetric operation, while encrypting a longer message (released gradients) would in theory require many asymmetric operations.

For example, for training MLP models (with 140100 parameters each of size 4 bytes) on MNIST dataset with the parameter upload rate and download rate of 100% in DSSGD, each party needs to send and download 140100 × 4 bytes = 0.56 Megabytes of gradients during each epoch. By contrast, in our DPPDL, during privacy-preserving collaborative deep learning, each party needs to send an encrypted message with two parts, namely the gradients encrypted by using a secure symmetric key encryption algorithm, and the encryption of the used symmetric key via a asymmetric key encryption scheme. As symmetric key encryption, such as AES-256, does not increase the size of the message (apart from potentially a few extra bytes for padding), the encrypted gradients in our scheme is of the same size (i.e., 0.56 MB). If we choose RSA-2048 for the asymmetric key encryption, we only introduce an additional 256-byte data (encrypted symmetric key) in the message, and this is negligible. Moreover, each party needs to send a very small amount of DPGAN samples as per local privacy level in both initial benchmarking and privacy-preserving collaborative deep learning, however the communication cost spent on transmitting samples is again negligible compared with communicating the encrypted gradients. Moreover, unlike DSSGD, the total communication cost in DPPDL is dependent on the number of parties n, because each party needs to send its encrypted gradients or DPGAN samples to the other $(n-1)$ parties, so that the communication cost is higher than a central server-based system, such as DSSGD. However, we remark that our DPPDL is well within the realm of practicality when the number of parties is not too large - as is the case for the collaboration among financial or biomedical institutions, while providing privacy and avoiding the inherent weaknesses in the server-based

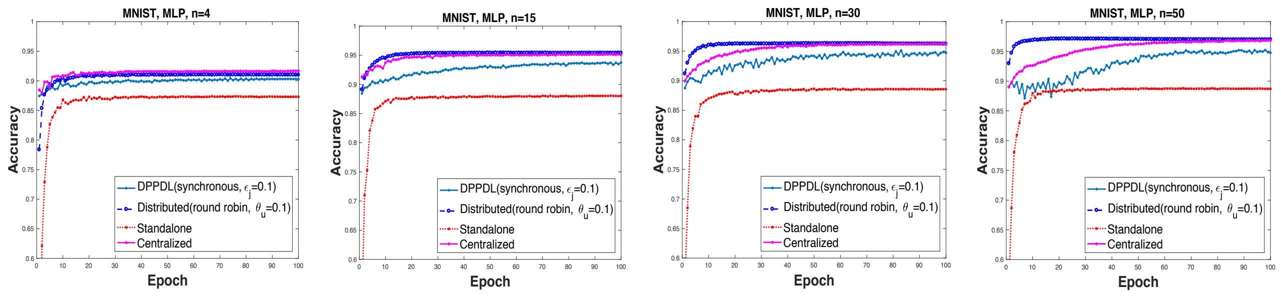

Fig. 6: MLP convergence over MNIST for different frameworks and different number of parties.

TABLE 5: Accuracy [%] over MNIST of varying party number settings, achieved by *Centralized*, *Standalone*, *Distributed* (DSSGD without DP, round robin, $\theta_u = 10\%$) and DPPDL (three settings as described in Section 5.3) frameworks using MLP and CNN architectures. P-k indicates there are k parties in the experiments.

Framework	MLP				CNN			
	P4	P15	P30	P50	P4	P15	P30	P50
Centralized	91.68	95.17	96.28	96.85	96.58	98.19	98.52	98.58
Distributed	91.67	95.17	96.33	97.35	96.25	98.04	98.63	98.83
Standalone	87.39	88.06	88.64	88.80	93.81	93.46	94.04	94.05
DPPDL (same privacy)	90.13	94.42	94.88	95.57	95.93	97.19	97.62	98.07
DPPDL (different privacy)	91.92	95.70	95.94	96.23	95.50	97.34	97.84	98.14
DPPDL (imbalanced partition)	90.75	94.37	94.75	95.21	95.23	97.50	97.82	98.22

TABLE 6: Accuracy [%] over SVHN of varying party number settings, achieved by *Centralized*, *Standalone*, *Distributed* (DSSGD without DP, round robin, $\theta_u = 10\%$) and DPPDL (three settings as described in Section 5.3) frameworks using MLP and CNN architectures. P-k indicates there are k parties in the experiments.

Framework	MLP				CNN			
	P4	P15	P30	P50	P4	P15	P30	P50
Centralized	75.40	83.08	85.77	87.15	90.50	91.88	93.42	95.44
Distributed	78.34	85.49	87.64	89.21	91.78	93.03	95.75	96.19
Standalone	57.85	58.77	57.90	59.18	80.24	80.74	81.29	81.60
DPPDL (same privacy)	73.74	82.55	84.86	86.51	90.07	91.18	92.74	94.83
DPPDL (different privacy)	74.16	82.67	85.25	86.57	89.91	91.15	92.59	95.18
DPPDL (imbalanced partition)	74.57	82.95	85.37	86.34	89.53	91.03	93.13	94.89

frameworks in Section 1.2. In terms of the computation cost spent on encryption, we tested the performance in a local computer with 2.7 GHz Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory. The time spent on the first layer of encryption using stream cipher is negligible, as modular addition (+) is very efficient [33]. The second layer of encrypting a 0.56 MB message with AES-256 takes about 9 ms, and the third layer of encrypting a 256-bit message using RSA (of key size 2048 bits) takes 4 ms, hence each party takes $\sim$ 13 ms on encryption of the message sent to any other party in the system, which is pretty fast.

6 Discussion

Data Augmentation and Collaboration. To facilitate credibility initialization, we apply data augmentation to expand local data size to ***help DPGAN generate reliable samples within a moderate privacy budget***. However, data augmentation is intended to increase the amount of training data using information inherent in local training data, and thus improve the generalizability of local model, while not helpful for generalizing to unseen data. In other words, it cannot represent global distribution, and this explains why parties still need collaboration for better utility even after data augmentation. By using DPGAN, it not only preserves privacy of the original data, but also preserves privacy of the augmented data that are similar to the original data.

Fairness and Privacy. With three-layer onion-style encryption, privacy is better preserved without compromising utility. We ensure fairness from two ways: (i) during initial benchmarking, parties generate DPGAN samples based on their local training data, which are then evaluated by other parties' standalone models to mutually initialize the local credibilities of other parties; and (ii) during collaborative learning process, each party randomly selects and shares a subset of DPGAN samples as per individual privacy level at each epoch of training, then updates the local credibilities of other parties who evaluate the received DPGAN samples using their local models at current epoch. Therefore, local credibility of each party keeps changing, reflecting more accurate contribution and thus possessing better fairness. Differentially private training of deep models provides another alternative solution by releasing gradients after each epoch of training, thus enabling each party to verify the claims of other parties and update their local credibilities as per the received gradients during collaborative learning process. One obstacle is that differentially private models may significantly reduce utility for small ϵ values.

Malicious Party Prevention. There might exist active adversaries who can perform two malicious behaviors as follows: 1) sending fake data to other parties; 2) stealing the exchanged information by eavesdropping on communication channels between honest participants. For the first malicious behavior, during initialization, this party may, for example, try to randomly sample from 10 classes as predicted labels for the received DPGAN samples, then release them to the corresponding party who publishes these DPGAN samples and requests labels. When the publisher receives the returned random labels from this party and detects that most of them are not aligned with the majority voting, *i.e.*, $\frac{m_j}{u_i} \ll c_{th}$, then this party will be reported as "low-quality contributor". If the majority of parties report one party as "low-quality contributor", then Blockchain rules out this party from the credible party set, and all parties would terminate the collaboration with this party. In this way, such malicious party is isolated from the beginning, and the collaboration among the remaining parties will not be affected. Even though the party might succeed in initialization somehow, the credibility of the malicious party is significantly lower compared with the other honest parties. To further detect and isolate the malicious party during the collaborative learning process, we repeat mutual evaluation at each epoch of collaborative learning by using samples generated at the initialization phase. Each party randomly selects and shares a subset of DPGAN samples as per individual privacy level at each epoch of collaborative learning, then updates the local credibilities of other parties by comparing the majority labels with the received labels output by the local models of other parties at current epoch of training. Hence, the chance of the survival of the malicious party is significantly reduced. Note that the threshold of acceptable lower bound of the quality can be agreed by the system according to its special need. For the second malicious behavior, our three-layer onion-style encryption scheme ensures communication security, preventing the success of eavesdropping attack. Therefore, our framework is robust and secure against these two types of active adversaries.

7 Related Work

Particularly relevant to this work is privacy and collaborative fairness. Since none of the previous works addressed the problem of **collaborative fairness** in deep learning, we only review the related work in privacy. Moreover, there is no privacy issues in standalone deep learning.

Privacy in centralized deep learning. As pointed out by Shokri *et al.* [11], centralized deep learning commonly comes with many privacy concerns. Specifically, all the sensitive training data are revealed to a susceptible third party; data owners have no control over the learning objective; the learned model is not directly available to data owners. To mitigate these privacy risks, Gilad-Bachrach *et al.* [17] developed CryptoNets to run deep learning on homomorphically encrypted data. However, CryptoNets assumes that neural network model has been trained beforehand, hence their system is mainly used to provide encrypted outputs to users [18]. Meanwhile, CryptoNets needs to alter the structure of neural networks and retrains them with special non-linear activation functions such as the square function to suit the computational needs for homomorphic encryption. This results in high computational costs and decreases in model accuracy. By contrast, SecureML [20] conducts privacy-preserving learning via *secure multiparty computation* (SMC), where data owners need to process, encrypt and/or secret-share their data among two non-colluding servers in the initial setup phase. SecureML allows data owners to train various models on their joint data without revealing any information beyond the outcome, however, at a cost of high computational and communication overhead, thereby decreasing interest in participation [39], [40].

Privacy in distributed deep learning. The most related work in distributed deep learning is *Distributed Selective Stochastic Gradient Descent* (DSSGD) introduced by Shokri *et al.* [11]. To preserve the shared model updates (gradients), instead of explicitly sharing training data, each party keeps its local model private, while iteratively updates its model by integrating differentially-private gradients from other parties through a central *parameter server* (PS). Communication cost is addressed by *Selective Stochastic Gradient Descent* (SSGD), where only a fraction (*e.g.*, 1%-10%) of local model gradients that are above a certain threshold or those with the largest absolute values are shared with the PS.

Each party computes and shares (with the PS) its local model gradients based on local training data, while updates its local model by downloading the most-updated parameters from the PS. However, their system requires a central parameter server to mediate training process, as illustrated in Fig. 1(c). Therefore, it suffers from the common issues in the central server-based frameworks as stated in Section 1.2. Moreover, extra risks are pointed out:

1: Meaningless differential privacy. The differential privacy bound is given per-parameter, but the large number of parameters prevents the technique from providing a meaningful privacy guarantee. For example, Shokri reports about 92% accuracy on SVHN with per-parameter privacy budget $\frac{\epsilon}{c} > 2$, where c is the number of the uploaded gradients of a model, which is over 300, 000 when the fraction of the uploaded gradients $\theta_u = 1$. Naively, this corresponds to a total $\epsilon > 600,000$ for $\theta_u = 1$ and $\epsilon > 60,000$ for $\theta_u = 0.1$. This will cause the privacy loss of a participant exceeds a meaningless large value of several thousands.

2: Privacy leakage. As evidenced in [16], local data information may be leaked to an honest-but-curious PS, even if only a small portion of local model updates is released to the PS. In particular, a PS can infer the truth value of the participates' data or the truth label with non-negligible probability for the local neural network with only one neuron. The above observations similarly hold for general neural networks, with both cross-entropy and squared-error cost functions. Even for general neural networks with regularization, the released local gradients can still reveal the truth value.

3: Vulnerability to active adversaries. Another assumption in their distributed framework is that all the parties are honest, neglecting the fact that some parties can take advantages of the system. In reality, if a party turns out to be malicious, it could easily sabotage the learning process by spoofing random samples or violate some of the privacy requirements by inferring information about the victim party's private data, which the attacker is not supposed to know. Hitaj *et al.* [41] described an active inference attack called *generative adversarial networks* (GAN) attack on deep neural networks in distributed deep learning. It exploits the real-time nature of the learning process that allows the adversarial party to train a GAN that generates prototypical samples of the targeted training set that was meant to be private and was intended to come from the same distribution as the training data. GAN attack makes distributed learning less desirable compared to the centralized learning. This is because, in centralized learning, only the server can violate the privacy of participating parties, but in distributed learning, any party can violate the privacy of other parties, even without attacking the server [41].

A special case of distributed deep learning is federated learning, which is tailored to deal with non-independent and identically distributed (non-IID), unbalanced and massively distributed data in mobile application. For example, FedAvg and FedSGD aim to train a shared global model while leaving the sensitive training data on users' mobile phones [12], [42], [43]. Similarly, in federated learning, to preserve privacy of individual model updates, Bonawitz *et al.* [42] proposed a practical secure aggregation protocol, which is proved to be secure in the honest-but-curious and active adversary settings, and the security is maintained even if an arbitrarily chosen subset of users drop out at any time. In particular, secure multiparty computation (SMC) is leveraged to compute sums of model parameter updates from individual users' devices in a secure manner, which comes at the cost of extra computation and communication overheads. Another more efficient method is to use differential privacy by enabling the server to add the tailored noise to the weighted-average user updates to guarantee user-level privacy [43]. However, the default trusted Google server is entitled to see all users' update clearly, aggregate individual updates and add noise to the aggregation, thus their method is not preferred when the server is untrusted.

Privacy in decentralized deep learning. In the first decentralized machine learning framework: ModelChain [15], Blockchain technology is integrated with privacy-preserving machine learning by incorporating the concept of boosting, *i.e.*, samples that are more difficult to classify are more likely to improve the model. To be more specific, the global model is initialized with the local model with the lowest error to prevent error propagation, and in the follow-up epochs, the party with the highest error is chosen to be the winner party to update the model as it contains the most information to further improve the model, and thus should be assigned a higher priority to be chosen as the party to update the model. The update process is repeated until the consensus global model is derived, that is, when a party wins the update bid in two continuous epochs. However, ModelChain is reasonable only if all the participants are completely honest as the winner party has the highest error, who can get access to all the intermediate models. Furthermore, ModelChain stated that privacy is preserved by exchanging zero patient data, however, the exchanged model-level information can still largely leak local data information [16].

8 Conclusion and Future Work

This paper proposes DPPDL, a decentralized privacy-preserving deep learning framework with fairness. Our enhanced framework shows the following properties: (1) it inherently resolves the relevant issues in the server-based frameworks, and investigates Blockchain for decentralization; (2) it makes the first investigation on the research problem of collaborative fairness in deep learning, by introducing a notion of local credibility and transaction points, which are initialized by initial benchmarking, and updated during privacy-preserving collaborative deep learning; (3) it combines *Differentially Private GAN* (DPGAN) and a three-layer onion-style encryption scheme to guarantee both accuracy and privacy; (4) it provides a viable solution to detect and reduce the impact of low-quality parties in the system. The experimental results demonstrate that our DPPDL achieves comparable accuracy to both the centralized and distributed selective SGD framework without differential privacy, and always delivers better results than the standalone framework, confirming the applicability of our proposed framework. A number of avenues for further work are attractive. In particular, we would like to study different malicious behaviours and explore real-world applications, such as fair and private collaboration among financial or biomedical institutions. We also expect to deploy our system on hardware in the near future.

Acknowledgments

Lingjuan Lyu is supported by an IBM PhD Fellowship in AI and Blockchain. The authors would like to thank Prof. Benjamin Rubinstein, Dr. Kumar Bhaskaran, and Prof. Marimuthu Palaniswami for their insightful discussions and supports on this work.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[4] Y. Wang, W. Liu, X. Ma, J. Bailey, H. Zha, L. Song, and S.-T. Xia, "Iterative learning with open-set noisy labels," *arXiv preprint arXiv:1804.00092*, 2018.

[5] X. Ma, Y. Wang, M. E. Houle, S. Zhou, S. M. Erfani, S.-T. Xia, S. Wijewickrema, and J. Bailey, "Dimensionality-driven learning with noisy labels," *arXiv preprint arXiv:1806.02612*, 2018.

[6] D. McGraw, "Building public trust in uses of health insurance portability and accountability act de-identified data," *Journal of the American Medical Informatics Association*, vol. 20, no. 1, pp. 29–34, 2013.

[7] R. Cummings, V. Gupta, D. Kimpara, and J. Morgenstern, "On the compatibility of privacy and fairness," 2019.

[8] M. Jagielski, M. Kearns, J. Mao, A. Oprea, A. Roth, S. Sharifi-Malvajerdi, and J. Ullman, "Differentially private fair learning," *arXiv preprint arXiv:1812.02696*, 2018.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[10] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, “Parallelized stochastic gradient descent,” in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[11] R. Shokri and V. Shmatikov, “Privacy-preserving deep learning,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1310–1321.

[12] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, “Federated learning of deep networks using model averaging,” *arXiv preprint arXiv:1602.05629*, 2016.

[13] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, “Bigchaindb: a scalable blockchain database,” *white paper, BigChainDB*, 2016.

[14] C. Fromknecht, D. Velicanu, and S. Yakoubov, “A decentralized public key infrastructure with identity retention.” *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014.

[15] L.-M. T-T Kuo, C-N Hsu, “Modelchain: Decentralized privacy-preserving healthcare predictive modeling framework on private blockchain networks,” in *ONC/NIST Blockchain in Healthcare and Research Workshop, Gaithersburg, MD*, September 26-7, 2016.

[16] 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.

[17] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” in *International Conference on Machine Learning*, 2016, pp. 201–210.

[18] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, “Crypto-nets: Neural networks over encrypted data,” *arXiv preprint arXiv:1412.6181*, 2014.

[19] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, “Oblivious multi-party machine learning on trusted processors.” in *USENIX Security Symposium*, 2016, pp. 619–636.

[20] P. Mohassel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 19–38.

[21] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[22] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[23] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford University Stanford, 2009, vol. 20, no. 09.

[24] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[25] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[26] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[27] P. Paillier *et al.*, “Public-key cryptosystems based on composite degree residuosity classes,” in *Eurocrypt*, vol. 99. Springer, 1999, pp. 223–238.

[28] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, “Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression,” *Journal of Cryptology*, vol. 31, no. 3, pp. 885–916, 2018.

[29] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[30] X. Zhang, S. Ji, and T. Wang, “Differentially private releasing via deep generative model,” *arXiv preprint arXiv:1801.01594*, 2018.

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

[32] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.

[33] C. Castelluccia, E. Mykletun, and G. Tsudik, “Efficient aggregation of encrypted data in wireless sensor networks,” in *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*. IEEE, 2005, pp. 109–117.

[34] L. Lyu, K. Nandakumar, B. Rubinstein, J. Jin, J. Bedo, and M. Palaniswami, “PPFA: Privacy preserving fog-enabled aggregation in smart grid,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3733–3744, 2018.

[35] C. De Canniere and B. Preneel, “Trivium,” in *New Stream Cipher Designs*. Springer, 2008, pp. 244–266.

[36] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, “Openpgp message format,” Tech. Rep., 2007.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[38] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

[39] L. Lyu, X. He, Y. W. Law, and M. Palaniswami, “Privacy-preserving collaborative deep learning with application to human activity recognition,” in *Proceedings of the 2017 ACM Conference on Information and Knowledge Management*. ACM, 2017, pp. 1219–1228.

[40] L. Lyu, J. C. Bezdek, X. He, and J. Jin, “Fog-embedded deep learning for the internet of things,” *IEEE Transactions on Industrial Informatics*, 2019.

[41] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, “Deep models under the gan: information leakage from collaborative deep learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 603–618.

[42] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[43] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” *arXiv preprint arXiv:1710.06963*, 2018.