Information Sciences 527 (2020) 108–127

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Privacy-Preserving distributed deep learning based on secret sharing

Jia Duan[a], Jiantao Zhou[a,*], Yuanman Li[b]

[a] State Key Laboratory of Internet of Things for Smart City, Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, China
[b] College of Electronics and Information Engineering, Shenzhen University, China

ARTICLE INFO

Article history:
Received 20 September 2019
Revised 21 March 2020
Accepted 23 March 2020
Available online 26 March 2020

Keywords:
Deep neural network
Distributed deep learning
Secure multi-party computation
Privacy preserving
Secret sharing

ABSTRACT

Distributed deep learning (DDL) naturally provides a privacy-preserving solution to enable multiple parties to jointly learn a deep model without explicitly sharing the local datasets. However, the existing privacy-preserving DDL schemes still suffer from severe information leakage and/or lead to significant increase of the communication cost. In this work, we design a privacy-preserving DDL framework such that all the participants can keep their local datasets private with low communication and computational cost, while still maintaining the accuracy and efficiency of the learned model. By adopting an effective secret sharing strategy, we allow each participant to split the intervening parameters in the training process into shares and upload an aggregation result to the cloud server. We can theoretically show that the local dataset of a particular participant can be well protected against the honest-but-curious cloud server as well as the other participants, even under the challenging case that the cloud server colludes with some participants. Extensive experimental results are provided to validate the superiority of the proposed secret sharing based distributed deep learning (SSDDL) framework.

1. Introduction

Recently, deep neural network (DNN) architectures have obtained impressive performance across a wide variety of fields, such as face recognition [32,37], machine translation [8,11], object detection [26,36], and object classification [14,19]. As the size of datasets increases, the computational intensity and memory demands of deep learning grow proportionally. Although in recent years significant advances have been made in GPU hardware, network architectures and training methods, the large-scale DNN training often takes an impractically long time on a single machine. Additionally, many accuracy improving strategies in the deep learning, such as scaling up the model parameters [31], utilizing complex model [9], and training on large-scale datasets [21], are also constrained by the computational power significantly.

Fortunately, distributed deep learning (DDL) framework provides a practicable and efficient solution to perform learning over large-scale datasets, especially when some datasets belong to different owners (and hence cannot be shared directly). To solve complex and time-consuming learning problems, DDL utilizes data parallelism and/or model parallelism [10]. In

* Corresponding author.
E-mail addresses: xuelandj@gmail.com (J. Duan), jtzhou@um.edu.mo (J. Zhou), yuanmanx.li@gmail.com (Y. Li).

https://doi.org/10.1016/j.ins.2020.03.074
0020-0255/

model parallelism, different participants in the distributed system are responsible for the computations in different parts (e.g. layers) of a single network. In data parallelism, different participants have a complete copy of the model, and each participant locally possesses a portion of the dataset. All the participants aim to learn an accurate deep model by sharing the training information (model parameters, gradients, etc.). These two kinds of parallelism strategies are not mutually exclusive. One can utilize model parallelism (model splits across GPUs) for each participant, and data parallelism among participants. Some representative DDL schemes are included in [10,18,22] and references therein.

Meanwhile, recently many privacy-preserving protocols have been suggested to leverage a public third-party to perform various computations, e.g., solving linear equations [7,42], linear programming [40,41,44], keyword search over outsourced data [15,39], cross-media retrieval [17,43], fair watermarking [46], nonnegative matrix factorization (NMF) [12], and deep learning [1,16,23,29,34] Among these various protocols, the ones concerning privacy-preserving deep learning are the most relevant to our work. According to the privacy issues of different phases in the network, these works can be grouped into two categories: one is to deal with privacy issues in the training phase [1,27,29,34,45,47,48] and the other is to preserve privacy in the testing phase [2,6,16,23]

Due to the confinement of the training data to its owner, it is natural to build a privacy-preserving deep learning framework based on the DDL architecture. The first privacy-preserving DDL was proposed in [34], where the gradients shared between the cloud server and participants are protected via a differential privacy mechanism. Unfortunately, Phong et al. [29] found that the shared gradients would cause severe information leakage of the local training data. To prevent the information leakage, they proposed to encrypt the shared gradients with additively homomorphic encryption (HE) [38]. More recently, Phong [28,30] devised a new privacy-preserving DDL framework by sharing encrypted network weights, rather than the encrypted gradients. Furthermore, Ma et al. [24] also suggested a privacy-preserving multi-party deep learning framework by incorporating ElGamal encryption [5] and Diffie-Hellman key exchange [35].

In this work, we propose a privacy-preserving SSDDL framework that enables multiple learning participants and one cloud server to collaboratively train an accurate DNN model with low communication and computational cost. To protect the input privacy, an effective secret sharing scheme is adopted. Specifically, instead of sharing gradients, each participant splits its own gradients into shares and distributes them to the other participants. Upon receiving the shares, each participant calculates an aggregation result, and uploads it to the cloud server for updating the global network parameters. It is shown theoretically that the local dataset of a particular participant can be well protected against the honest-but-curious cloud server as well as the other participants, even under the challenging case that the cloud server colludes with some participants. Extensive experimental results are provided to validate the superiority of the proposed SSDDL framework.

The rest of this paper is organized as follows. Section 2 reviews and analyzes the state-of-the-art privacy-preserving DDL frameworks. Some preliminaries are given in Section 3. Our proposed privacy-preserving SSDDL framework is presented in Section 4. In Section 5, we provide detailed security analysis for evaluating our proposed scheme, under the designed security experiments. Section 6 offers the experimental results and finally we conclude in Section 7.

2. Review and analyses of the privacy-preserving DDL frameworks

In this section, before giving our analyses of the existing privacy-preserving DDL frameworks, we first provide the details of the state-of-art frameworks.

2.1. Review of the privacy-preserving DDL frameworks

Deep learning aims to extract complex features from high-dimensional data and use them to build a model which can map inputs to outputs. Usually, deep learning architectures are constructed as multi-layer networks so that more abstract features are computed as nonlinear functions of lower-level ones. A traditional multi-layer DNN architecture can be built up by one input layer, one output layer and l hidden layers, as illustrated in Fig. 1. Each hidden layer is connected with the output of the previous layer. For simplicity, we consider a single layer of the DNN, and the expression can be extended to the other layers as well. Let $\mathbf{x} \in \mathbb{R}^{1\times u}$ denote an input vector, and $\mathbf{h} \in \mathbb{R}^{1\times v}$ denotes the output of the layer. Then, the computation of a single layer can be expressed as

$$\mathbf{h} = f(\mathbf{x} * \mathbf{W} + \mathbf{b}), \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{u\times v}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^{1\times v}$ is the row vector of biases, and f denotes the activation function such as sigmoid, rectified linear unit (ReLU), softplus, hyperbolic tangent, max-out, etc.

The learning task is considered as an optimization problem, which determines these weight variables ($\mathbf{W}$ and $\mathbf{b}$) by minimizing a pre-defined cost function over a training dataset. The cost function is evaluated over all the data in the training dataset, or over a subset (mini-batch). In practice, stochastic gradient descent (SGD), which computes the gradient over an extremely small subset of the whole dataset, is a commonly used technique for solving the optimization problem of deep learning. Let $\mathbf{\Theta}$ be the flattened vector of all the weights variables (consisting of $\mathbf{W}$ and $\mathbf{b}$ in each layer) of the deep network. Then, the update rule of SGD can be formulated as:

$$\mathbf{\Theta} := \mathbf{\Theta} + \mathbf{G}, \tag{2}$$

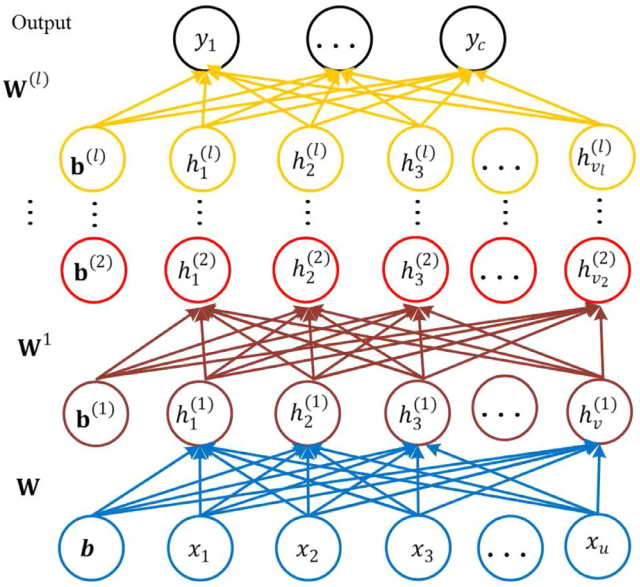


Fig. 1. The traditional architecture of deep neural network.

where $\mathbf{G} = -\alpha \cdot \hat{\mathbf{G}}$ describes the parameter change, $\hat{\mathbf{G}}$ denotes the gradients of the parameters, and α represents the learning rate. The learning rate α can be fixed or adaptively determined, as presented in [13]. It should be also noted that each weight variable in $\boldsymbol{\Theta}$ is updated independently from the others.

Based on the SGD in the traditional deep learning, Shokri et al. [34] proposed a privacy-preserving DDL framework, called distributed selective stochastic gradient descent (DSSGD). Specifically, several participants collaboratively train a deep model with a honest-but-curious cloud server. The architecture of the DNN is fixed and known for all the participants and the cloud server. Each participant trains the model on its own dataset, and shares the gradients with the cloud server. The cloud server is responsible to maintain the global parameters (model parameters), and share them to the participants. Inspired by the selective SGD, each participant uploads a fraction of gradients, rather than all the gradients, so that the communication cost can be lowered without affecting the model accuracy. In addition, Shokri et al. [34] also suggested to protect the input data privacy by adopting differential privacy strategy, at the cost of degraded model accuracy.

Although DSSGD was claimed to be able to provide excellent efficiency, accuracy, and data privacy, Phong et al. [29] recently pointed out its unsatisfactory privacy issues. It was observed that the gradients of the first layer can completely leak the information of the input data. Hence, the gradients shared in the plaintext between the cloud server and participants would cause severe information leakage of the local training data, defeating the ultimate goal of privacy-preserving. To remedy this drawback, Phong et al. [29] also suggested a simple countermeasure by encrypting the shared gradients with additively HE. Specifically, all the participants negotiate a pair of public key *pk* and secret key *sk* for performing additively HE. The secret key *sk* is kept confidential from the cloud server, but needs to be shared among all the participants. Upon each round of local training, each participant employs HE to encrypt the gradients with the public key *pk*, and sends the encrypted version to the cloud server for updating the model parameters by resorting to (2) in the encrypted domain. By using the secret key *sk*, the participants can easily decrypt the global parameters. Under such framework, it was claimed that the model parameters and the gradients can be made private from the cloud server. Furthermore, Phong et al. [30] proposed a privacy-preserving DNN by sharing weights, instead of gradients. Specifically, they presented two kinds of network architectures: Server-aided Network Topology (SNT) and Fully-connected Network Topology (FNT). In a SNT system, similar to [29], a symmetric key is shared among all the participants, but is kept secret to the cloud server. Each participant trains the weights over its own dataset locally and encrypts the weights with the symmetric key. The encrypted weights are then uploaded to the cloud server for synchronization. In FNT system, without a centralized cloud server, all the participants train and transfer their weights one by one in a fixed order. Due to the absence of the cloud server, the symmetric encryption is abandoned.

Bonawitz et al. [4] also designed a failure-robust protocol which can privately share the gradients between the participants and the cloud server. Specifically, each participant adds double masks to the gradient and uploads the masked gradient to the cloud server. Upon receiving enough masked gradients, the cloud server aggregates them and removes the masks by the corresponding keys. Sharmir's t-out-of-n secret sharing scheme is adopted to handle the dropped participants during the interactions. Under such framework, it was proved that the joint view of the server and any set of $t-1$ participants cannot leak any information about the inputs of the other participants.

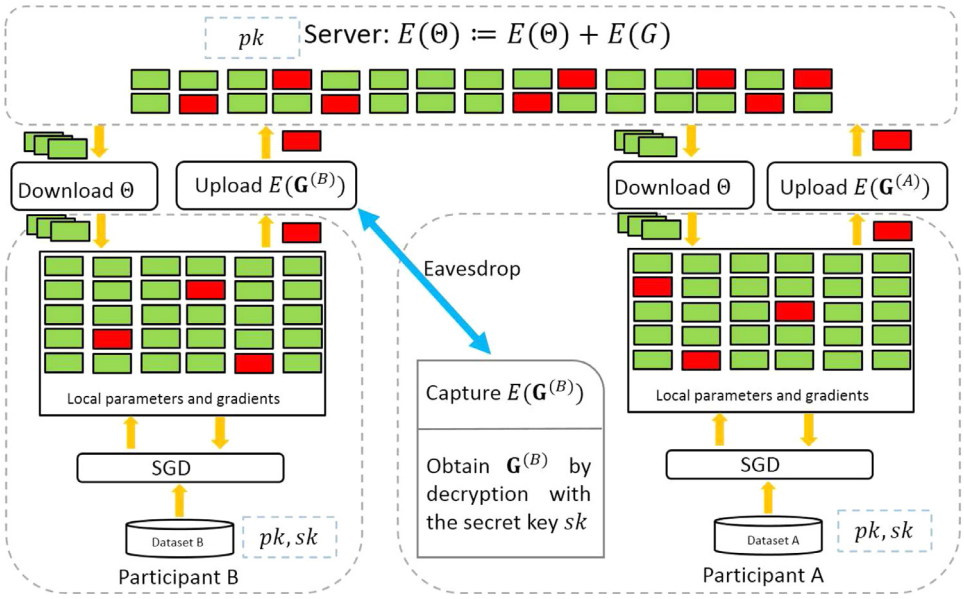


Fig. 2. The privacy issues in the DDL framework.

2.2. Some additional analyses on the existing privacy-preserving DDL schemes

As described above, the previous works [29] and [34] provided two efficient privacy-preserving DDL frameworks where the input (local dataset) was protected against the cloud server. However, privacy issues still exist in these frameworks.

Specifically, Phong et al. [29]. found that the k-th component of the input data $\mathbf{x} = \{x_1, x_2, \cdots, x_u\}$ can be recovered by the gradients of the k-th weight variable $\mathbf{W}_k$ and the bias $\mathbf{b}$ in the first layer. Furthermore, in the strategy [34], the gradient of the weight variable $\mathbf{W}_k$ in the first layer can be considered as proportional to the input x_k. In this case, the adversary can utilize the gradients to easily produce a related input by scaling. Hence, sharing the gradients in plain is insecure in the DDL framework.

In the work [29], although the additively HE provided the confidentiality of the gradients against the cloud server, the computational cost was dramatically increased. Furthermore, the scheme cannot protect the privacy of the local dataset against the other participants. For instance, there is a participant A who is curious about the local dataset of the participant B. Assume that the participant A can eavesdrop the communication channel between the cloud server and the participant B. As shown in Fig. 2, the participants A and B choose a pair of key $\{pk, sk\}$ for additively HE scheme. The secret key sk is kept confidential from the cloud server, but is known to the participants A and B. After the local training, the participants A and B encrypt their parameter changes $\mathbf{G}^{(A)}$ and $\mathbf{G}^{(B)}$ by additively HE respectively. It can be expressed as

$$E(\mathbf{G}^{(A)}) = HE(pk, \mathbf{G}^{(A)}), \qquad E(\mathbf{G}^{(B)}) = HE(pk, \mathbf{G}^{(B)}), \tag{3}$$

where $HE(pk, \mathbf{G}^{(A)})$ represents that the parameter change $\mathbf{G}^{(A)}$ of the participant A is encrypted by the additively HE with the public key pk. When the participant B uploads the encrypted $E(\mathbf{G}^{(B)})$ to the cloud server, the participant A may capture the encrypted $E(\mathbf{G}^{(B)})$ by eavesdropping the communication channel. Since the secret key sk is known to all the participants, the participant A can easily obtain the parameter change $\mathbf{G}^{(B)}$ of the participant B by decrypting with the secret key sk. As the parameter change $\mathbf{G}^{(B)}$ is the product of the learning rate and the gradient vector, the participant A can produce a proportional gradient vector by scaling. In this case, the participant A can derive the sensitive information of the dataset located in the participant B from the parameter change $\mathbf{G}^{(B)}$. Similar information leakage would occur in the schemes presented in [28,30]. Though Phong et al. [30] provided the analysis showing that their schemes are secure against the cloud server colluding with the participants, the gradients information still leaks to the other participants, due to the same symmetric key employed.

Even though each participant establishes a TSL/SSL secure channel with the cloud server, the participants' parameter changes still can leak to the cloud server if the server colludes with only one participant. Specifically, the cloud server can obtain the secret key sk from the colluding participant, and easily derives the sensitive information of other participants from the decrypted parameter changes. Furthermore, employing the TSL/SSL secure channel will increase the communication cost. For instance, the model structure and the global parameters will be individually sent to each participant, implying that it increases the communication cost compared with the broadcasting strategy.

The frameworks in [28–30] are essentially based on the fact that all the participants employed the same public and secret key pair. Now, we discuss a more challenging situation that the participants employ different key pairs. Assume that

the participant A and the participant B utilize different key pairs $\{pk_A, sk_A\}$ and $\{pk_B, sk_B\}$ in the DDL framework, respectively. Then, the encrypted parameter change of participant A and B can be expressed as

$$E(\mathbf{G}^{(A)}) = HE(pk_A, \mathbf{G}^{(A)}), \qquad E(\mathbf{G}^{(B)}) = HE(pk_B, \mathbf{G}^{(B)}). \tag{4}$$

Due to the different encryption key pairs, the addition operation over the encrypted gradients $E(\mathbf{G}^{(A)})$ and $E(\mathbf{G}^{(B)})$ cannot be performed correctly. Consequently, the global parameters cannot be updated accordingly, making the scheme utilizing different key pairs infeasible. These above problems motivate us to design and implement a low-complexity privacy-preserving deep learning framework, without leaking the local datasets to both the cloud server and the other participants.

3. Preliminaries

In this section, we introduce some cryptographic primitives needed for our proposed framework.

3.1. Secret sharing scheme

Secret sharing refers to methods for distributing a secret among a group of participants, each of whom is allocated a share of the secret. Shamir's t-out-of-n secret sharing scheme [33], as a classical and widely used method, allows a user to split a secret s into n shares, such that any t shares can reconstruct the secret s, but any $t-1$ or less shares leaves s completely undetermined. Specifically, the scheme is composed of two algorithms: sharing algorithm $\mathsf{S.share}()$ and reconstruction algorithm $\mathsf{S.recon}()$. The algorithms can be described as follows:

- $\mathsf{S.share}(s, t, n) \rightarrow \{s_1, s_2, \cdots, s_n\}$: Given the threshold t, the number of shares n, and the secret s, it produces a set of shares $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$.
- $\mathsf{S.recon}(\mathcal{M}, t) \rightarrow s$: Given a subset $\mathcal{M}$ of the total shares $\mathcal{S}$, where $\mathcal{M} \subseteq \mathcal{S}$, and $|\mathcal{M}| \geq t$, it reconstructs the secret s from shares.

3.2. ϵ-Forward Secure pseudorandom number generator

Before giving the description of the ϵ-forward secure pseudorandom number generator (PRNG), we first present the definition of the standard PRNG. Let G: $\{0,1\}^{\lambda} \rightarrow \{0,1\}^{p(\lambda)}$, $p(\lambda) > \lambda$, be a family of deterministic polynomial time computable functions. If its output is computationally indistinguishable from true randomness, then G is called a PRNG [3].

Compared with the standard PRNG described above, a ϵ-forward secure PRNG is a stateful PRNG [3]. It is an iterative and stateful algorithm, which produces some output bits as pseudorandom number and updates the seed at each invocation. It can be described as $G_{fs} : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{\lambda} \times \{0,1\}^{l_1}$. At the period i, the input bit string $e_i \in \{0,1\}^{\lambda}$ is considered as the current state. The next state $e_{i+1} \in \{0,1\}^{\lambda}$ and the pseudorandom number $r_i \in \{0,1\}^{l_1}$ are regarded as the output of the period i. A ϵ-forward secure PRNG can be easily constructed with a standard PRNG $G : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{\lambda+l_1}$ by splitting its output into the next state and the actual output. Specifically, it can be done by setting $G_{fs}(e_i) = G_0(e_i)||G_1(e_i)$, where $G_0(e_i) = e_{i+1} \in \{0,1\}^{\lambda}$ and $G_1(e_i) = r_i \in \{0,1\}^{l_1}$. Then G_{fs} is a ϵ-forward secure PRNG with G_0 as the next state and G_1 as the pseudorandom output of the current state [3]. In the sequel, we use the notation $G^i_{fs}(e_1)$ to denote the output pseudorandom random sequence $(r_1, r_2, \ldots, r_i)$ up to the period i with the seed e_1. Namely, we have

$$G^i_{fs}(e_1) = G_1(e_1)||G_1(e_2)||\cdots||G_1(e_i). \tag{5}$$

Then, ϵ-forward secure PRNG can be formally defined as follows [3]:

Definition 1. Let $G_{fs} : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{\lambda} \times \{0,1\}^{l_1}$ be a PRNG. If the initial state (seed) K is chosen uniformly at random from $\{0,1\}^{\lambda}$, then at any period i, for any probabilistic polynomial time algorithm $\mathcal{D}$, which outputs 1 or 0 as a distinguisher, the probability of distinguishing the output from a truly random sequence is at most ϵ. It can be expressed as

$$\left|\Pr[\mathcal{D}(G^i_{fs}(K)) = 1] - \Pr[\mathcal{D}(\pi) = 1]\right| \leq \epsilon, \tag{6}$$

where ϵ is a negligible function related to the seed length λ. The sequence π consists of a truly random sequence $(r'_1, r'_2, \ldots, r'_i)$, with each element randomly chosen from $\{0,1\}^{l_1}$.

The definition indicates that it is negligibly possible to distinguish the sequence output from a truly random one by a ϵ-forward secure PRNG.

4. The proposed privacy-preserving SSDDL framework

In this section, we provide the details of our proposed privacy-preserving SSDDL framework, starting with the descriptions on the design goals and system model.

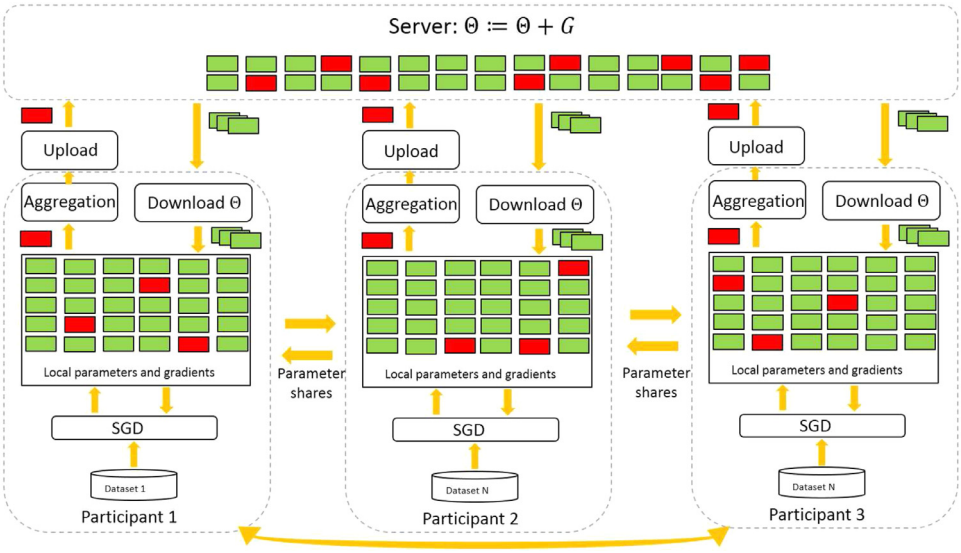


Fig. 3. An example of 3 participants in privacy-preserving distributed deep neural network.

4.1. Design goals

To provide the privacy protection and practicability of the proposed framework, we identify the following three design goals.

- **Privacy**: The local dataset should be kept secret against both the cloud server and the other participants. Any adversaries cannot learn or guess the meaningful information concerning the local dataset. More formal security definitions will be given in Section 5.
- **Efficiency**: The computational complexity of each participant should be less than that of a centralized model, in which a single server runs the deep learning over the entire dataset. Also, the communication cost among the cloud server and the participants should be minimized.
- **Accuracy**: The accuracy of the final model should be identical or close to the one if the network is trained over the aggregate dataset in a single server.

4.2. System model

We consider a privacy-preserving SSDDL framework with one cloud server and N participants. Without loss of generality, we assume that the number of participants $N \geq 3$, which is primarily due to the secret sharing strategy to be adopted. Also, we assume that the framework is operated under the honest-but-curious setting. It implies that the cloud server and all the participants follow the specifications, but are curious about the information of the local datasets of the other participants. Specifically, the cloud server maintains the model parameters $\mathbf{\Theta} = \{\theta_1, \theta_2, \cdots, \theta_n\}$, where n is the total number of the parameters in the network. Each participant P_i possesses a private local dataset $\mathbf{D}_i$, where $1 \leq i \leq N$. All the local datasets form the entire training data $\mathbf{D} = \bigcup_{i=1}^{N} \mathbf{D}_i$. The goal of all the participants is to obtain an accurate model, which is close to or even identical to the one trained over the entire $\mathbf{D}$, under the privacy-preserving constraint, i.e., the local training data should be kept private.

Fig. 3 presents the key modules of the system with 3 participants as an example. First of all, the cloud server initializes the network randomly and shares the model replicas to each participant. Each participant P_i then executes the local training using the local dataset $\mathbf{D}_i$, based on the global model parameters $\mathbf{\Theta}$ obtained from the cloud server. Upon the local training, each participant obtains the parameter change $\mathbf{G}^{(i)}$ according to the update rule (2). As mentioned previously, $\mathbf{G}^{(i)}$ cannot be shared in the plaintext. To protect the privacy of $\mathbf{G}^{(i)}$, each participant P_i splits it into shares and distributes the shares to the other participants. Here, the shares of the parameter change $\mathbf{G}^{(i)}$ are generated through a simple yet effective secret sharing scheme, which will be detailed in the subsections below. After collecting shares from all the other participants, each participant P_i aggregates them and uploads the aggregation result to the cloud server. Eventually, the cloud server, upon receiving all the aggregation results from participants, updates the global parameters accordingly. The above procedures are repeated iteratively until the model converges.

Although the strategy in [4] and our framework both employ the secret sharing scheme, the technical intuitions are different. As presented in Section 2, Bonawitz et al. [4] protected the gradients by double masking, and utilized secret

sharing to handle the dropped participants. In our proposed framework, the m-out-of-m secret sharing scheme is adopted to split the gradients into shares. Additionally, Our proposed framework only needs the splitting shares function, while not the reconstruction function. In other words, upon receiving the aggregated shares from participants, the cloud server does not require to reconstruct the specific gradient from the shares.

The proposed privacy-preserving SSDDL framework consists of the following five key modules:

- $\mathsf{Init}(\mathbf{P}, m) \rightarrow (\mathbf{\Theta}, \mathbf{\Gamma})$: The cloud server builds up a DNN and initializes the model parameter $\mathbf{\Theta}$ with random values. On input the set of the participants $\mathbf{P}$ and group size m, the cloud server partitions all the participants into groups and outputs the group set $\mathbf{\Gamma}$.
- $\mathsf{SelectGroup}(\mathbf{\Gamma}) \rightarrow \gamma$: On input the group set $\mathbf{\Gamma}$, the cloud server chooses one group γ for the collaborative training.
- $\mathsf{LocalTrain}(\mathbf{\Theta}, \gamma, m, \eta) \rightarrow (\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(m)})$: On input the global parameters $\mathbf{\Theta}$, the group γ, the group size m and the uploading rate η, each participant P_i in the group γ runs the deep learning algorithm locally on his own dataset $\mathbf{D}_i$. Then each participant P_i outputs its own parameter change $\mathbf{G}^{(i)}$. The details will be given later in Section 4.5.
- $\mathsf{ParaShare}(\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(m)}, \lambda) \rightarrow (\mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(m)})$: On input the security parameter λ, each participant P_i splits its own parameter change $\mathbf{G}^{(i)}$ and sends the shares to the other participants in the same group γ. After collecting different shares from the other participants, each participant P_i outputs an aggregation result $\mathbf{C}^{(i)}$ and uploads it to the cloud server. The details of generating and distributing shares will be provided in the subsequent section.
- $\mathsf{ParaUpdate}(\mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(m)}) \rightarrow \mathbf{\Theta}$: Upon receiving the results $\{\mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(m)}\}$ from the participants in the group γ, the cloud server generates the updated global parameters of the DNN.

The modules SelectGroup, LocalTrain, ParaShare, ParaUpdate will be repeated sequentially until the deep model converges or a maximum iteration is achieved.

In the following, we provide the details concerning the five parts of our proposed privacy-preserving SSDDL framework.

4.3. $\mathsf{Init}(\mathbf{P}, \mathsf{m}) \rightarrow (\mathbf{\Theta}, \mathbf{\Gamma})$

The cloud server builds up the DNN, and initializes the model parameters $\mathbf{\Theta} = \{\theta_1, \theta_2, \cdots, \theta_n\}$ with some random values. Then the cloud server broadcasts $\mathbf{\Theta}$ to all the participants. To reduce the communication cost among the participants, the cloud server partitions all the participants into several groups. Specifically, all the participants in $\mathbf{P}$ are divided into $\lceil N/m \rceil$ groups of size m, where N represents the number of the participants in $\mathbf{P}$. Here, we assume $m \geq 3$, which is a requirement raised from the secret sharing strategy to be adopted.

4.4. $\mathsf{SelectGroup}(\mathbf{\Gamma}) \rightarrow \gamma$

The cloud server selects one group γ in the group set $\mathbf{\Gamma}$ for the following collaborative training. The selection criterion could be *random* or *sequential*. With *random* order, the group γ in the group set $\mathbf{\Gamma}$ is uniformly selected at random. With *sequential* order, the group γ is selected one by one in a fixed order. After selecting a group γ, the cloud server broadcasts the global parameters $\mathbf{\Theta}$ and the group vector γ to all the participants in the group γ. By accessing the group vector γ, each participant in the group γ knows the membership of γ. All the participants in this group γ interact with the cloud server to perform the following operations. The rest of the participants will wait until the interaction is completed.

4.5. $\mathsf{LocalTrain}(\mathbf{\Theta}, \gamma, \mathsf{m}, \eta) \rightarrow (\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(\mathsf{m})})$

Upon receiving the global parameters $\mathbf{\Theta}$, each participant P_i in the group γ replaces the previously trained local parameters with $\mathbf{\Theta}$. Then, each participant P_i, for $1 \leq i \leq m$, runs the deep learning algorithm on his own dataset $\mathbf{D}_i$ with the parameters $\mathbf{\Theta}$ obtained from the cloud server. Specifically, each participant P_i runs SGD algorithm for n_{epo} epochs to obtain the trained parameters $\mathbf{\Theta}^{(i)}$. Here, the number of epochs n_{epo} influences the convergence rate of the model, which will be discussed with experiments in Section 6. Let the vector $\mathbf{G}^{(i)}$ denote the parameter change after the local training over the dataset $\mathbf{D}_i$, which can be formulated as

$$\mathbf{G}^{(i)} = \mathbf{\Theta}^{(i)} - \mathbf{\Theta}. \tag{7}$$

Inspired by the selective SGD, the participant P_i chooses a fraction η of $\mathbf{G}^{(i)}$ to be shared, rather than sharing the entire $\mathbf{G}^{(i)}$. The main intuition behind this strategy is that some parameters contribute much more to the DNN's cost function. Furthermore, some features of the input data are more important than the others. Generally, there are two parameter selection criteria. One is to select those parameters who have larger parameter changes. The other method is randomly sampling. To be consistent with the settings in [34], we randomly select a fraction η of $\mathbf{G}^{(i)}$ to be shared. Let $\mathcal{X}$ be the indexes of the $\eta \times n$ values in vector $\mathbf{G}^{(i)}$. Then we keep these $\eta \times n$ entries in $\mathbf{G}^{(i)}$ unchanged and set the others to 0, namely,

$$\mathbf{G}_k^{(i)} = \begin{cases} \mathbf{G}_k^{(i)} & k \in \mathcal{X} \\ 0 & \text{otherwise}, \end{cases} \tag{8}$$

where $\mathbf{G}_k^{(i)}$ is the k-th component of $\mathbf{G}^{(i)}$.

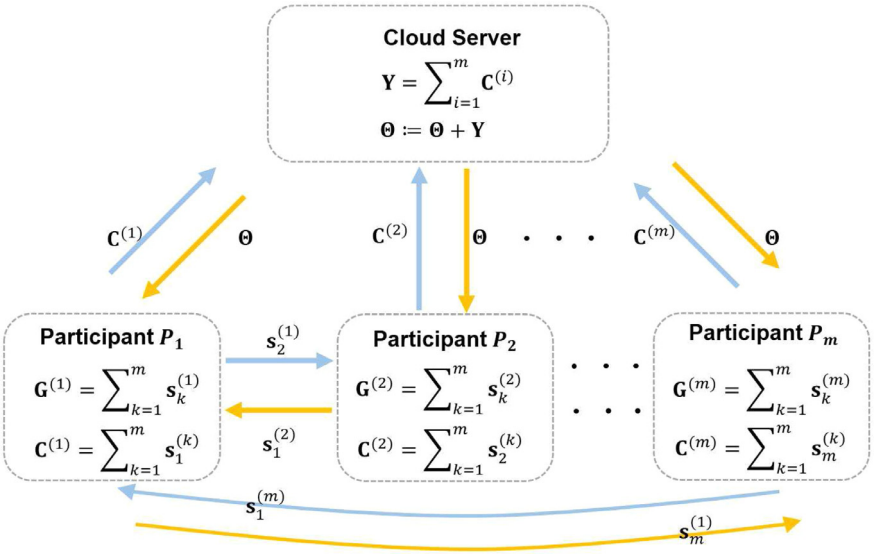


Fig. 4. The architecture of distributed deep neural network.

4.6. $\mathsf{ParaShare}(\mathbf{G}^{(1)}, \cdots, \mathbf{G}^{(m)}, \lambda) \rightarrow (\mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(m)})$

After the local training, each participant P_i in the group γ needs to share its parameter change $\mathbf{G}^{(i)}$ to the cloud server for updating the global parameters. However, as analyzed in Section 2, the parameter change $\mathbf{G}^{(i)}$ may leak the input information to the cloud server and the other participants. Hence, each participant P_i has to share the parameter change $\mathbf{G}^{(i)}$ in a private way, rather than uploading it directly to the cloud server in the plaintext. A straightforward strategy is to encrypt the parameter change $\mathbf{G}^{(i)}$ with homomorphic encryption method and execute the addition over the encrypted domain on the cloud server. However, homomorphic encryption increases the computational/communication cost dramatically. Further, due to the same secret key used among all the participants, the parameter change $\mathbf{G}^{(i)}$ will also be leaked to the other participants.

To tackle the above problems, in this work, we propose to use a simple yet effective m-out-of-m additive secret sharing scheme to protect the parameter change $\mathbf{G}^{(i)}$. The main idea behind our strategy is that each participant splits the parameter change into shares, and uploads the aggregation result to the cloud server. Since the cloud server only requires a summation of all the parameter change $\mathbf{G}^{(i)}$ to update the global parameter $\mathbf{\Theta}$, there is no need to recover or reconstruct the specific value of the secret parameter change $\mathbf{G}^{(i)}$ from its shares. Hence, compared with the traditional secret sharing scheme, our proposed framework only needs the sharing algorithm $\mathsf{S.share}()$, while not the reconstruction algorithm $\mathsf{S.recon}()$. In other words, the cloud server can update the global parameters, but cannot recover $\mathbf{G}^{(i)}$ from the received information (aggregation results). All the computations are carried out over the cyclic group $\mathbb{Z}_p$ with order p. We first encode all the parameter changes $\{\mathbf{G}^{(i)}\}_{i=1}^{m}$ into the range of $\mathbb{Z}_p$. We utilize the binary representation of a fixed precision number to convert a floating number into an integer.

The procedure of generating the parameter shares consists of the following four parts, as can be shown in Fig. 4. For each participant P_i in the group γ, it sequentially performs:

- $\mathsf{Setup}(1^\lambda) \rightarrow K^{(i)}$: Given the security parameter λ, the participant P_i generates a secret key $K^{(i)}$ from a specified key space $\{0, 1\}^\lambda$.
- $\mathsf{GenShares}(K^{(i)}, \mathbf{G}^{(i)}, m) \rightarrow (\mathbf{s}_1^{(i)}, \cdots, \mathbf{s}_m^{(i)})$: The participant P_i utilizes the secret key $K^{(i)}$ to drive a ϵ-forward secure PRNG so as to generate a pseudorandom number sequence.
  Specifically, on input a seed $K^{(i)}$, a ϵ-forward secure PRNG can produce a pseudorandom sequence $\mathbf{r} = \{r_i\}_{i=1}^{t}$, where $t = n \times (m-1)$ and $r_i \in \{0,1\}^{l_1}$. We then employ this sequence $\mathbf{r}$ to generate shares of the parameter change $\mathbf{G}^{(i)}$ as described in Algorithm 1. On input a secret vector $\mathbf{G}^{(i)}$, the pseudorandom sequence $\mathbf{r}$ and the number of shares m, it outputs m shares of the input vector.

$$\mathbf{Alg}(\mathbf{G}^{(i)}, \mathbf{r}, m) \rightarrow \{\mathbf{s}_1^{(i)}, \mathbf{s}_2^{(i)}, \cdots, \mathbf{s}_m^{(i)}\} \in \mathbb{Z}_p^{n \times m}, \tag{9}$$

  where **Alg** denotes the Algorithm 1, and n represents the length of each vector. According to the generation procedure, it can be easily derived that

$$\sum_{k=1}^{m} \mathbf{s}_k^{(i)} = \mathbf{G}^{(i)} \tag{10}$$

- $\mathsf{Dist}(\mathbf{s}_1^{(i)}, \cdots, \mathbf{s}_{i-1}^{(i)}, \mathbf{s}_{i+1}^{(i)}, \cdots, \mathbf{s}_m^{(i)})$: The participant P_i keeps the share $\mathbf{s}_i^{(i)}$, and distributes the other shares $\mathbf{s}_j^{(i)}$ to the participant P_j, respectively, where $1 \le i, j \le m$ and $i \ne j$.
- $\mathsf{Compute}(\mathbf{s}_i^{(1)}, \cdots, \mathbf{s}_i^{(m)}) \to \mathbf{C}^{(i)}$: Upon receiving $m-1$ shares $s_i^{(k)}$, where $k = 1, \cdots, i-1, i+1, \cdots, m$, from the other participants, the participant P_i computes the aggregation result of the corresponding shares as

$$\mathbf{C}^{(i)} = \sum_{k=1}^{m} \mathbf{s}_i^{(k)} \tag{11}$$

Then, P_i sends the result $\mathbf{C}^{(i)}$ to the cloud server for the subsequent update procedure.

4.7. $\mathsf{ParaUpdate}(\mathbf{C}^{(1)}, \cdots, \mathbf{C}^{(m)}) \to \mathbf{\Theta}$

Upon receiving all the aggregation results $\mathbf{C}^{(i)}$, $1 \le i \le m$, from all the participants, the cloud server updates the global parameters by

$$\mathbf{\Theta} := \mathbf{\Theta} + \mathbf{Y}, \tag{12}$$

where

$$\mathbf{Y} = \sum_{i=1}^{m} \mathbf{C}^{(i)}. \tag{13}$$

Here, $\mathbf{Y}$ is essentially equivalent to the summation of all the parameter change $\mathbf{G}^{(i)}$ from each participant P_i, for $1 \le i \le m$. Specifically, it can be derived from the Eqs. (10) and (11)

$$\mathbf{Y} = \sum_{i=1}^{m} \mathbf{C}^{(i)} = \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{s}_i^{(j)} = \sum_{i=1}^{m} \mathbf{G}^{(i)}. \tag{14}$$

After updating the global parameters, the cloud server completes the interaction with the group γ. Then the cloud server repeats SelectGroup module to choose a new group for the collaborative training. Recall that the selection criterion could be *random* or *sequential*. Then the participants in the new group interact with the cloud server to train the model parameters. In other words, the four modules SelectGroup, LocalTrain, ParaShare, ParaUpdate will be repeated sequentially until the deep model converges or a maximum iteration is achieved.

5. Security experiments and analysis

In this section, we theoretically analyze our proposed framework under the honest-but-curious setting. Before giving the security analysis, we first describe the security experiments, based on which we then present the security analysis. We use the notation $x \xleftarrow{R} \mathcal{S}$ to denote that x is chosen uniformly at random from the set $\mathcal{S}$. A function $f(n)$ is said to be negligible if for a sufficiently large n, its value is smaller than the inverse of any polynomial $poly(n)$.

5.1. Security experiments

The security requirement of privacy includes the dataset privacy and parameter privacy, meaning that the privacy of the local dataset and the intervening parameters during the collaborative learning process should be protected. In our framework, since the local dataset has never been transferred out of its local domain, we focus on considering the potential information leakage of the intervening parameters during the learning process. In general, the adversary could be the cloud server and/or the participants. Hence, we describe the security experiments in three cases: privacy against the participants; privacy against the cloud server; and privacy against the cloud server colluding with some participants. In the following, we present the security experiment of each case in details.

Firstly, let us discuss the security experiment with honest-but-curious participants. Under this scenario, all the participants follow the system specification; but each participant attempts to obtain or recover the other participants' local datasets from the intervening parameters. Since each participant P_i performs the same operations to protect its local parameter change $\mathbf{G}^{(i)}$, we only need to analyze one participant, and the result is applicable to the other participants as well. For simplicity, we consider the participant P_1 as the challenger, who aims to protect its local parameter change $\mathbf{G}^{(1)}$ from the other participants' attack. In the experiment, the adversary $\mathcal{A}$ is allowed to query the participant P_1 on its input $\mathbf{G}_j^{(1)}$ of its choice a polynomial times, and obtain the corresponding public shares $\tilde{\mathbf{S}}_j^{(1)} = \{\mathbf{s}_2, \mathbf{s}_3, \cdots, \mathbf{s}_m\}$, where j is the index for the query. Essentially, this is a type of chosen plaintext attack (CPA). $\mathcal{A}$ outputs two inputs $\mathbf{G}_{[0]}^{(1)}$ and $\mathbf{G}_{[1]}^{(1)}$ on which it would like

to be challenged. A random bit b is drawn. $\mathcal{A}$ is then given $\tilde{\mathbf{S}}_{[b]}^{(1)}$, which is the public shares of $\mathbf{G}_{[b]}^{(1)}$. Eventually, the adversary $\mathcal{A}$ outputs its guess for b and wins if it could correctly determine the value of b, i.e., distinguishing between $\mathbf{G}_{[0]}^{(1)}$ and $\mathbf{G}_{[1]}^{(1)}$.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-P}}(\lambda, m)$

$K_1^{(1)} \leftarrow \mathsf{Setup}(1^\lambda), \quad \mathbf{G}_1^{(1)} \xleftarrow{R} \mathbb{Z}_p^n$
$\mathbf{S}_1^{(1)} \triangleq \{\mathbf{s}_1, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{GenShares}(K_1^{(1)}, \mathbf{G}_1^{(1)}, m)$
$\tilde{\mathbf{S}}_1^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}, \quad \mathsf{Dist}(\tilde{\mathbf{S}}_1^{(1)})$
for $j = 2$ **to** q do
  $\mathbf{G}_j^{(1)} \leftarrow \mathcal{A}(\tilde{\mathbf{S}}_1^{(1)}, \tilde{\mathbf{S}}_2^{(1)}, \cdots, \tilde{\mathbf{S}}_{j-1}^{(1)}), \quad K_j^{(1)} \leftarrow \mathsf{Setup}(1^\lambda)$
  $\mathbf{S}_j^{(1)} \triangleq \{\mathbf{s}_1, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{GenShares}(K_j^{(1)}, \mathbf{G}_j^{(1)}, m)$
  $\tilde{\mathbf{S}}_j^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}, \quad \mathsf{Dist}(\tilde{\mathbf{S}}_j^{(1)})$
$\{\mathbf{G}_{[0]}^{(1)} \in \mathbb{Z}_p^n, \mathbf{G}_{[1]}^{(1)} \in \mathbb{Z}_p^n\} \leftarrow \mathcal{A}(\tilde{\mathbf{S}}_1^{(1)}, \tilde{\mathbf{S}}_2^{(1)}, \cdots, \tilde{\mathbf{S}}_q^{(1)})$
$b \xleftarrow{R} \{0, 1\}, K \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathbf{S}_{[b]}^{(1)} \triangleq \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{GenShares}(K, \mathbf{G}_{[b]}^{(1)}, m)$
$\tilde{\mathbf{S}}_{[b]}^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}, \quad \mathsf{Dist}(\tilde{\mathbf{S}}_{[b]}^{(1)})$
$\hat{b} \leftarrow \mathcal{A}(\tilde{\mathbf{S}}_1^{(1)}, \tilde{\mathbf{S}}_2^{(1)}, \cdots, \tilde{\mathbf{S}}_q^{(1)}, \tilde{\mathbf{S}}_{[b]}^{(1)})$
if $\hat{b} = b$ return 1, else return 0

For any $\lambda \in \mathbb{N}$, we define the advantage of an adversary $\mathcal{A}$ making at most $q = poly(\lambda)$ queries in the above security game as

$$\mathbf{Adv}_{\mathcal{A}}^{\text{CPA-P}}(\lambda, m, q) = \left| \Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-P}}(\lambda, m) = 1] - 1/2 \right|. \tag{15}$$

Definition 2. We say that a privacy-preserving DDL scheme ensures data secrecy against any honest-but-curious participants if for any adversary $\mathcal{A}$ and λ, it holds that the advantage $\mathbf{Adv}_{\mathcal{A}}^{\text{CPA-P}}(\lambda, m, q)$ is negligible.

Secondly, we would like to discuss the security experiment with honest-but-curious cloud server, meaning that the cloud server follows the framework specification; but attempts to recover the datasets of the participants from the intervening parameters. Similar to the first security experiment, we consider the participant P_1 as the challenger, who aims to protect its local parameter change $\mathbf{G}^{(1)}$ from the cloud server's attack. In the experiment, the adversary $\mathcal{A}$ is allowed to query the participant P_1 on its input $\mathbf{G}_j^{(1)}$ of its choice a polynomial times, to obtain the corresponding aggregation results $\mathbf{C}_j^{(1)}$, where j is the index for the query. $\mathcal{A}$ outputs two inputs $\mathbf{G}_{[0]}^{(1)}$ and $\mathbf{G}_{[1]}^{(1)}$ on which it would like to be challenged. A random bit b is drawn. $\mathcal{A}$ is then given $\mathbf{C}_{[b]}^{(1)}$, which is the aggregation result from P_1. Eventually, the adversary $\mathcal{A}$ outputs its guess for b and wins if it could correctly determine the value of b, i.e., distinguishing between $\mathbf{G}_{[0]}^{(1)}$ and $\mathbf{G}_{[1]}^{(1)}$.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m)$

$\{\mathbf{G}_1^{(1)}, \cdots, \mathbf{G}_1^{(m)}\} \xleftarrow{R} \mathbb{Z}_p^{n \times m}$
$\{\mathbf{C}_1^{(1)}, \cdots, \mathbf{C}_1^{(m)}\} \leftarrow \mathsf{ParaShare}(\mathbf{G}_1^{(1)}, \cdots, \mathbf{G}_1^{(m)}, \lambda)$
$\tilde{\mathbf{C}}_1 \triangleq \{\mathbf{C}_1^{(1)}, \cdots, \mathbf{C}_1^{(m)}\}$
for $j = 2$ **to** q do
  $\mathbf{G}_j^{(1)} \leftarrow \mathcal{A}(\tilde{\mathbf{C}}_1, \cdots, \tilde{\mathbf{C}}_{j-1}), \quad \{\mathbf{G}_j^{(2)}, \cdots, \mathbf{G}_j^{(m)}\} \xleftarrow{R} \mathbb{Z}_p^{n \times (m-1)}$
  $\{\mathbf{C}_j^{(1)}, \cdots, \mathbf{C}_j^{(m)}\} \leftarrow \mathsf{ParaShare}(\mathbf{G}_j^{(1)}, \cdots, \mathbf{G}_j^{(m)}, \lambda)$
  $\tilde{\mathbf{C}}_j \triangleq \{\mathbf{C}_j^{(1)}, \cdots, \mathbf{C}_j^{(m)}\}$
$\{\mathbf{G}_{[0]}^{(1)} \in \mathbb{Z}_p^n, \mathbf{G}_{[1]}^{(1)} \in \mathbb{Z}_p^n\} \leftarrow \mathcal{A}(\tilde{\mathbf{C}}_1, \cdots, \tilde{\mathbf{C}}_q)$
$b \xleftarrow{R} \{0, 1\}, \{\mathbf{G}_{[b]}^{(2)}, \cdots, \mathbf{G}_{[b]}^{(m)}\} \xleftarrow{R} \mathbb{Z}_p^{n \times (m-1)}$
$\{\mathbf{C}_{[b]}^{(1)}, \cdots, \mathbf{C}_{[b]}^{(m)}\} \leftarrow \mathsf{ParaShare}(\mathbf{G}_{[b]}^{(1)}, \cdots, \mathbf{G}_{[b]}^{(m)}, \lambda)$
$\tilde{\mathbf{C}}_{[b]} \triangleq \{\mathbf{C}_{[b]}^{(1)}, \cdots, \mathbf{C}_{[b]}^{(m)}\}$
$\hat{b} \leftarrow \mathcal{A}(\tilde{\mathbf{C}}_1, \cdots, \tilde{\mathbf{C}}_q, \tilde{\mathbf{C}}_{[b]})$
if $\hat{b} = b$ return 1, else return 0

For any $\lambda \in \mathbb{N}$, we define the advantage of an adversary $\mathcal{A}$ making at most $q = poly(\lambda)$ queries in the above security game as

$$\mathbf{Adv}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m, q) = \left|\Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m) = 1] - 1/2\right|. \tag{16}$$

Definition 3. We say that a privacy-preserving DDL scheme ensures data secrecy against the honest-but-curious cloud server if for any adversary $\mathcal{A}$ and λ, it holds that the advantage $\mathbf{Adv}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m, q)$ is negligible.

In the above experiments, we analyze the security games with honest-but-curious server or participants. It implies that there is no collusion between the participants and the cloud server. Now we present the security experiment for a more challenging case: the honest-but-curious cloud server colludes with some participants. Specifically, we utilize $\tilde{\mathbf{P}}$ to denote the set of curious participants in a group γ that collude with the cloud server, and let $\hat{\mathbf{P}}$ be the set of trustful participants in γ that have no intention to collude with the cloud server. Clearly, we have $\gamma = \tilde{\mathbf{P}} \cup \hat{\mathbf{P}}$ and $m = |\hat{\mathbf{P}}| + |\tilde{\mathbf{P}}|$. The cloud server colluding with the participants in $\tilde{\mathbf{P}}$ attempts to recover or obtain the local datasets of the participants in $\hat{\mathbf{P}}$. Here, it should be noted that the security experiment becomes identical to $\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m)$ when $|\tilde{\mathbf{P}}| = 0$. In the experiment, we assume that $P_1 \in \hat{\mathbf{P}}$ is a challenger, who aims to protect its local parameter change $\mathbf{G}^{(1)}$ from the cloud server and the participants in $\tilde{\mathbf{P}}$. The adversary $\mathcal{A}$ is allowed to query the participant P_1 on its input $\mathbf{G}_j^{(1)}$ of its choice a polynomial times, to obtain the corresponding results $\mathbf{C}_j^{(1)}$, where j is the index for query. $\mathcal{A}$ outputs two inputs $\mathbf{G}_{[0]}^{(1)}$ and $\mathbf{G}_{[1]}^{(1)}$ on which it would like to be challenged. A random bit b is drawn. $\mathcal{A}$ is then given $\mathbf{C}_{[b]}^{(1)}$, which is the aggregation result of participant P_1. Eventually, the adversary $\mathcal{A}$ outputs its guess for b and wins if it could correctly determine the value of b, i.e., distinguish between $\mathbf{G}_{[0]}^{(1)}$ and $\mathbf{G}_{[1]}^{(1)}$. Since the query is almost the same as that of the experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m)$, in the below, we only provide the procedures after q times queries of the participant P_1 on its input for simplicity.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{Mixed}}(\lambda, m)$

$\{\mathbf{G}_{[0]}^{(1)}, \mathbf{G}_{[1]}^{(1)} \in \mathbb{Z}_p^n\} \leftarrow \mathcal{A}(\tilde{\mathbf{C}}_1, \cdots, \tilde{\mathbf{C}}_q, \{\mathbf{S}_1^{(k)}, \cdots, \mathbf{S}_q^{(k)}\}_{P_k \in \tilde{\mathbf{P}}})$
$b \overset{R}{\leftarrow} \{0, 1\}$, $\{\mathbf{G}_{[b]}^{(2)}, \cdots, \mathbf{G}_{[b]}^{(m)}\} \overset{R}{\leftarrow} \mathbb{Z}_p^{n \times (m-1)}$
for each P_i in group γ, $i \in [1, m]$ do
  $K^{(i)} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathbf{r}^{(i)} \triangleq \{r_1^{(i)}, \cdots, r_t^{(i)}\} \leftarrow G_{fs}^t(K^{(i)})$
  $\mathbf{S}_{[b]}^{(i)} \triangleq \{\mathbf{s}_{1,[b]}^{(i)}, \cdots, \mathbf{s}_{m,[b]}^{(i)}\} \leftarrow \mathsf{Alg}(\mathbf{G}_{[b]}^{(i)}, \mathbf{r}^{(i)}, m)$
  $\tilde{\mathbf{S}}_{[b]}^{(i)} \triangleq \{\mathbf{s}_{2,[b]}^{(i)}, \cdots, \mathbf{s}_{m,[b]}^{(i)}\}$, $\mathsf{Dist}(\tilde{\mathbf{S}}_{[b]}^{(i)})$
  $\mathbf{C}_{[b]}^{(i)} \leftarrow \mathsf{Compute}(\mathbf{s}_{i,[b]}^{(1)}, \cdots, \mathbf{s}_{i,[b]}^{(m)})$
$\tilde{\mathbf{C}}_{[b]} \triangleq \{\mathbf{C}_{[b]}^{(1)}, \cdots, \mathbf{C}_{[b]}^{(m)}\}$
$\hat{b} \leftarrow \mathcal{A}(\tilde{\mathbf{C}}_1, \cdots, \tilde{\mathbf{C}}_q, \tilde{\mathbf{C}}_{[b]}, \{\mathbf{S}_1^{(k)}, \cdots, \mathbf{S}_q^{(k)}, \mathbf{S}_{[b]}^{(k)}\}_{P_k \in \tilde{\mathbf{P}}})$
if $\hat{b} = b$ return 1, else return 0

5.2. Security analysis

To formulate the data secrecy in our proposed framework, we proceed with security games and analyze the adversary $\mathcal{A}$'s advantage in winning the experiment.

Theorem 1. *The proposed privacy-preserving SSDDL framework holds indistinguishability with honest-but-curious participants according to the Definition 2.*

Proof. The proof is based on the analysis of $\mathcal{A}$'s advantage $\mathbf{Adv}_{\mathcal{A}}^{\text{CPA-P}}(\lambda, m, q)$.

Game $\mathbf{Game_0}$. Define $\mathbf{Game_0}$ to be the same as $\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-P}}(\lambda, m)$. Specifically, $\mathcal{A}$ wins the game $\mathbf{Game_0}$ if the security experiment returns 1, implying that the adversary can distinguish $\mathbf{G}_{[0]}^{(1)}$ from $\mathbf{G}_{[1]}^{(1)}$. When $\mathbf{G}_{[b]}^{(1)}$ is given, where $b \in \{0, 1\}$ is a random bit, the participant P_1 generates $\mathbf{S}_{[b]}^{(1)} \triangleq \{\mathbf{s}_1, \cdots, \mathbf{s}_m\}$ by an effective m-out-of-m secret sharing scheme which is driven by a ϵ-forward secure PRNG. We now construct a detailed version of the security game, aiming to distinguish $\mathbf{G}_{[0]}^{(1)}$ from $\mathbf{G}_{[1]}^{(1)}$ according to $\mathbf{S}_{[b]}^{(1)}$.

Game $\mathbf{Game_1}$. Define $\mathbf{Game_1}$ to be as below.

$K_1^{(1)} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathbf{G}_1^{(1)} \xleftarrow{R} \mathbb{Z}_p^n$, $\mathbf{r}_1 \triangleq \{r_1, \cdots, r_t\} \leftarrow G_{fs}^t(K_1^{(1)})$
$\mathbf{S}_1^{(1)} \triangleq \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{Alg}(\mathbf{G}_1^{(1)}, \mathbf{r}_1, m)$
$\bar{\mathbf{S}}_1^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}$, $\mathsf{Dist}(\bar{\mathbf{S}}_1^{(1)})$
for $j = 2$ **to** q do
  $\mathbf{G}_j^{(1)} \leftarrow \mathcal{A}(\bar{\mathbf{S}}_1^{(1)}, \bar{\mathbf{S}}_2^{(1)}, \cdots, \bar{\mathbf{S}}_{j-1}^{(1)})$, $K_j^{(1)} \leftarrow \mathsf{Setup}(1^\lambda)$
  $\mathbf{r}_j \triangleq \{r_1, \cdots, r_t\} \leftarrow G_{fs}^t(K_j^{(1)})$
  $\mathbf{S}_j^{(1)} \triangleq \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{Alg}(\mathbf{G}_j^{(1)}, \mathbf{r}_j, m)$
  $\bar{\mathbf{S}}_j^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}$, $\mathsf{Dist}(\bar{\mathbf{S}}_j^{(1)})$
$\{\mathbf{G}_{[0]}^{(1)} \in \mathbb{Z}_p^n, \mathbf{G}_{[1]}^{(1)} \in \mathbb{Z}_p^n\} \leftarrow \mathcal{A}(\bar{\mathbf{S}}_1^{(1)}, \bar{\mathbf{S}}_2^{(1)}, \cdots, \bar{\mathbf{S}}_q^{(1)})$
$b \xleftarrow{R} \{0, 1\}$, $K \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathbf{r} \triangleq \{r_1, \cdots, r_t\} \leftarrow G_{fs}^t(K)$
$\mathbf{S}_{[b]}^{(1)} \triangleq \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{Alg}(\mathbf{G}_{[b]}^{(1)}, \mathbf{r}, m)$
$\bar{\mathbf{S}}_{[b]}^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}$, $\mathsf{Dist}(\bar{\mathbf{S}}_{[b]}^{(1)})$
$\hat{b} \leftarrow \mathcal{A}(\bar{\mathbf{S}}_1^{(1)}, \bar{\mathbf{S}}_2^{(1)}, \cdots, \bar{\mathbf{S}}_q^{(1)}, \bar{\mathbf{S}}_{[b]}^{(1)})$
if $\hat{b} = b$ return 1, else return 0

Here, Alg denotes the Algorithm 1, and G_{fs}^t is a ϵ-forward secure PRNG described in Section 4.

Algorithm 1 Generating shares of the input vector.

Require: Secret vector $\mathbf{q} = \{q_1, q_2, \cdots, q_n\} \in \mathbb{Z}_p^n$, the number of shares m, pseudorandom number sequence $\mathbf{r} = \{r_i\}_{i=1}^t$, where $t = n \times (m-1)$, maximum of pseudorandom number $\mathsf{RMAX} = 2^{l_1} - 1$, maximum ZMAX and minimum ZMIN of $\mathbb{Z}_p$.
Ensure: Shares $\{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \in \mathbb{Z}_p^{n \times m}$.
1: Generate m empty vectors $\{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \in \mathbb{Z}_p^{n \times m}$.
2: **for** $d = 1$ to t **do**
3: Generate a random integer $h_d \in \mathbb{Z}_p$ by setting $h_d = \mathsf{ZMIN} + \lfloor \frac{r_d}{\mathsf{RMAX}+1} \cdot (\mathsf{ZMAX} - \mathsf{ZMIN} + 1) \rfloor$.
4: **end for**
5: **for** $i = 1$ to n **do**
6: **for** $j = 1$ to $m - 1$ **do**
7: Put the random integer into the i-th element of $\mathbf{s}_j$ by setting $\mathbf{s}_j[i] = h_{n \times (i-1)+j}$.
8: **end for**
9: **end for**
10: Set $\mathbf{s}_m = \mathbf{q} - \sum_j^{m-1} \mathbf{s}_j$ // element-wise substraction
11: **return** Shares $\{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\}$.

Let T_1 be the event that $\hat{b} = b$ in the game **Game$_1$**. It can be easily seen that the probability of distinguishing $\mathbf{G}_{[0]}^{(1)}$ from $\mathbf{G}_{[1]}^{(1)}$ is equal to $\Pr[T_1]$.

Game **Game$_2$**. Define **Game$_2$** to be the same as **Game$_1$** except that the sequence $\mathbf{r} = \{r_1, \cdots, r_t\}$ is replaced by a truly random one. Let T_2 be the event that $\hat{b} = b$ in the game **Game$_2$**. Due to the employment of the truly random sequence, the adversary's output $\hat{b}$ is independent of the hidden bit b. Hence, the adversary $\mathcal{A}$ can correctly guess the bit b with probability 1/2, i.e.,

$$\Pr[T_2] = \frac{1}{2}. \tag{17}$$

Comparing the two games **Game$_1$** and **Game$_2$**, the difference lies in the input of Alg. Let $\mathcal{D}$ be a distinguishing algorithm, which has queried q times of participant P_1 on its input, working as follows.

Algorithm $\mathcal{D}(\tilde{r}_1, \tilde{r}_2, \cdots, \tilde{r}_t)$
$\{\mathbf{G}_{[0]}^{(1)} \in \mathbb{Z}_p^n, \mathbf{G}_{[1]}^{(1)} \in \mathbb{Z}_p^n\} \leftarrow \mathcal{A}(\bar{\mathbf{S}}_1^{(1)}, \bar{\mathbf{S}}_2^{(1)}, \cdots, \bar{\mathbf{S}}_q^{(1)})$
$b \xleftarrow{R} \{0, 1\}$
$\mathbf{S}_{[b]}^{(1)} \triangleq \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\} \leftarrow \mathsf{Alg}(\mathbf{G}_{[b]}^{(1)}, \tilde{r}_1, \cdots, \tilde{r}_t, m)$
$\bar{\mathbf{S}}_{[b]}^{(1)} \triangleq \{\mathbf{s}_2, \cdots, \mathbf{s}_m\}$, $\mathsf{Dist}(\bar{\mathbf{S}}_{[b]}^{(1)})$
$\hat{b} \leftarrow \mathcal{A}(\bar{\mathbf{S}}_1^{(1)}, \bar{\mathbf{S}}_2^{(1)}, \cdots, \bar{\mathbf{S}}_q^{(1)}, \bar{\mathbf{S}}_{[b]}^{(1)})$
if $b = \hat{b}$ output 1, else output 0

With the above distinguishing algorithm $\mathcal{D}$, we can see that

$$\Pr[T_1] = \Pr[\mathcal{D}(G_{fs}^t(K)) = 1]. \tag{18}$$

Similarly, we have

$$\Pr[T_2] = \Pr[\mathcal{D}(r'_1, \cdots, r'_t) = 1], \tag{19}$$

where $(r'_1, \cdots, r'_t)$ is a truly random sequence with each element being randomly selected from $\{0,1\}^{l_1}$. Then we can infer that $\left|\Pr[T_1] - \Pr[T_2]\right|$ is equivalent to the adversary's advantage for ϵ-forward secure PRNG, i.e.,

$$\begin{aligned}\left|\Pr[T_1] - \Pr[T_2]\right| &= \left|\Pr[\mathcal{D}(G^t_{fs}(K)) = 1] - \Pr[\mathcal{D}(r'_1, \cdots, r'_t) = 1]\right| \\ &\leq \epsilon,\end{aligned} \tag{20}$$

where the last inequality holds from (6). Then, combining the equality (17) and (20), we have

$$\left|\Pr[T_1] - 1/2\right| \leq \epsilon, \tag{21}$$

where ϵ is considered as negligible. It implies that the probability of distinguishing $\mathbf{G}^{(1)}_{[0]}$ from $\mathbf{G}^{(1)}_{[1]}$ is arbitrarily close to 1/2. Therefore, we conclude that the adversary's advantage is negligible in wining the game $\mathbf{Game_0}$. This completes the proof. □

Theorem 2. *The proposed privacy-preserving SSDDL framework holds indistinguishability with honest-but-curious cloud server according to the Definition 3.*

Proof. The proof is based on the analysis of $\mathcal{A}$'s advantage $\mathbf{Adv}^{\text{CPA-CS}}_{\mathcal{A}}(\lambda, m, q)$. Similar to the proof of Theorem 1, we can also define security games to analyze the adversary's advantage.

Game $\mathbf{Game_0}$. Define $\mathbf{Game_0}$ to be the same as $\mathbf{Exp}^{\text{CPA-CS}}_{\mathcal{A}}(\lambda, m)$. Specifically, $\mathcal{A}$ wins the game if the experiment returns 1, indicating that the adversary $\mathcal{A}$ can distinguish $\mathbf{G}^{(1)}_{[0]}$ from $\mathbf{G}^{(1)}_{[1]}$. When $\mathbf{G}^{(1)}_{[b]}$ is given, where $b \in \{0, 1\}$ is a random bit, the participant P_1 produces an aggregation result $\mathbf{C}^{(1)}_{[b]}$ and uploads it to the cloud server. Here, the generation procedure of $\mathbf{C}^{(1)}_{[b]}$ is driven by a ϵ-forward secure PRNG. We now construct a detailed version of the security game.

Game $\mathbf{Game_1}$. Define $\mathbf{Game_1}$ in the pseudocode above. For simplicity, we only expand the procedures after q times queries of the participant P_1 on its input. Let T_1 be the event that $\hat{b} = b$ in the game $\mathbf{Game_1}$. It can be easily seen that the probability of distinguishing $\mathbf{G}^{(1)}_{[0]}$ from $\mathbf{G}^{(1)}_{[1]}$ is equal to $\Pr[T_1]$.

Experiment $\mathbf{Exp}^{\text{CPA-CS}}_{\mathcal{A}}(\lambda, m)$

$\{\mathbf{G}^{(1)}_1, \cdots, \mathbf{G}^{(m)}_1\} \xleftarrow{R} \mathbb{Z}_p^{n\times m}$
$\{\mathbf{C}^{(1)}_1, \cdots, \mathbf{C}^{(m)}_1\} \leftarrow \mathsf{ParaShare}(\mathbf{G}^{(1)}_1, \cdots, \mathbf{G}^{(m)}_1, \lambda)$
$\bar{\mathbf{C}}_1 \triangleq \{\mathbf{C}^{(1)}_1, \cdots, \mathbf{C}^{(m)}_1\}$
for $j = 2$ **to** q do
  $\mathbf{G}^{(1)}_j \leftarrow \mathcal{A}(\bar{\mathbf{C}}_1, \cdots, \bar{\mathbf{C}}_{j-1}), \quad \{\mathbf{G}^{(2)}_j, \cdots, \mathbf{G}^{(m)}_j\} \xleftarrow{R} \mathbb{Z}_p^{n\times(m-1)}$
  $\{\mathbf{C}^{(1)}_j, \cdots, \mathbf{C}^{(m)}_j\} \leftarrow \mathsf{ParaShare}(\mathbf{G}^{(1)}_j, \cdots, \mathbf{G}^{(m)}_j, \lambda)$
  $\bar{\mathbf{C}}_j \triangleq \{\mathbf{C}^{(1)}_j, \cdots, \mathbf{C}^{(m)}_j\}$
$\{\mathbf{G}^{(1)}_{[0]} \in \mathbb{Z}_p^n, \mathbf{G}^{(1)}_{[1]} \in \mathbb{Z}_p^n\} \leftarrow \mathcal{A}(\bar{\mathbf{C}}_1, \cdots, \bar{\mathbf{C}}_q)$
$b \xleftarrow{R} \{0, 1\}$, $\{\mathbf{G}^{(2)}_{[b]}, \cdots, \mathbf{G}^{(m)}_{[b]}\} \xleftarrow{R} \mathbb{Z}_p^{n\times(m-1)}$
for each P_i in group γ, $i \in [1, m]$ do
  $K^{(i)} \leftarrow \mathsf{Setup}(1^\lambda), \quad \mathbf{r}^{(i)} \triangleq \{r^{(i)}_1, \cdots, r^{(i)}_t\} \leftarrow G^t_{fs}(K^{(i)})$
  $\mathbf{S}^{(i)}_{[b]} \triangleq \{\mathbf{s}^{(i)}_{1,[b]}, \cdots, \mathbf{s}^{(i)}_{m,[b]}\} \leftarrow \mathsf{Alg}(\mathbf{G}^{(i)}_{[b]}, \mathbf{r}^{(i)}, m)$
  $\bar{\mathbf{S}}^{(i)}_{[b]} \triangleq \{\mathbf{s}^{(i)}_{2,[b]}, \cdots, \mathbf{s}^{(i)}_{m,[b]}\}, \quad \mathsf{Dist}(\bar{\mathbf{S}}^{(i)}_{[b]})$
  $\mathbf{C}^{(i)}_{[b]} \leftarrow \mathsf{Compute}(\mathbf{s}^{(1)}_{i,[b]}, \cdots, \mathbf{s}^{(m)}_{i,[b]})$
$\bar{\mathbf{C}}_{[b]} \triangleq \{\mathbf{C}^{(1)}_{[b]}, \cdots, \mathbf{C}^{(m)}_{[b]}\}$
$\hat{b} \leftarrow \mathcal{A}(\bar{\mathbf{C}}_1, \cdots, \bar{\mathbf{C}}_q, \bar{\mathbf{C}}_{[b]})$
if $\hat{b} = b$ return 1, else return 0

where Alg denotes the Algorithm 1, and G^t_{fs} represents a ϵ-forward secure PRNG described in Section 4.

Game $\mathbf{Game_2}$. Define $\mathbf{Game_2}$ to be the same as $\mathbf{Game_1}$ except that the sequence $\mathbf{r}^{(1)} = \{r^{(1)}_1, \cdots, r^{(1)}_t\}$ is replaced by a truly random one. Let T_2 be the event that $\hat{b} = b$ in the game $\mathbf{Game_2}$. Due to the employment of the truly random sequence, the adversary's output $\hat{b}$ is independent of the hidden bit b. Hence, the adversary $\mathcal{A}$ can correctly guess the bit b with probability 1/2, i.e.,

$$\Pr[T_2] = \frac{1}{2}. \tag{22}$$

Similar to the analysis in the proof of Theorem 1, we can infer that $\left|\Pr[T_1]-\Pr[T_2]\right|$ is equivalent to the adversary's advantage for ϵ-forward secure PRNG, i.e.,

$$\begin{aligned}\left|\Pr[T_1]-\Pr[T_2]\right| &= \left|\Pr[\mathcal{D}(G_{fs}^{t}(K^{(1)}))=1]-\Pr[\mathcal{D}(r_1',\cdots,r_t')=1]\right| \\ &\leq \epsilon,\end{aligned} \tag{23}$$

where the last inequality holds from (6). Then, combining (22) and (23), we have

$$\left|\Pr[T_1]-1/2\right| \leq \epsilon, \tag{24}$$

where ϵ is considered as negligible. It implies that the probability of distinguishing $\mathbf{G}_{[0]}^{(1)}$ from $\mathbf{G}_{[1]}^{(1)}$ is arbitrarily close to 1/2. Therefore, we conclude that the adversary's advantage is negligible in wining the game $\mathbf{Game_0}$. This completes the proof. □

Theorem 3. *The proposed privacy-preserving SSDDL framework protects the privacy of the local dataset from the honest-but-curious cloud server and the colluding participants if and only if the number of trustful participants* $|\hat{\mathbf{P}}| \geq 2$.

Proof. The proof is based on the analysis of the adversary's advantage in the experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{Mixed}}(\lambda, m)$. Similar to the proof of Theorem 2, we can also define security games to analyze the adversary's advantage. For simplicity, we only provide a proof sketch.

Firstly, we consider the distinguishing probability of the adversary when $|\hat{\mathbf{P}}| \geq 2$. Comparing to the security experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{CPA-CS}}(\lambda, m)$, the adversary $\mathcal{A}$ can also obtain the information of the participants in the set $\tilde{\mathbf{P}}$. Specifically, we consider an extreme case when $|\hat{\mathbf{P}}| = 2$. Without loss of generality, we assume that $\hat{\mathbf{P}} = \{P_1, P_2\}$, and the rest of participants $\tilde{\mathbf{P}} = \{P_3, \cdots, P_m\}$ collude with the cloud server. The adversary has the knowledge of the aggregation results $\{\tilde{\mathbf{C}}_1, \cdots, \tilde{\mathbf{C}}_q, \tilde{\mathbf{C}}_{[b]}\}$ and the shares $\{\mathbf{S}_1^{(k)}, \cdots, \mathbf{S}_q^{(k)}, \mathbf{S}_{[b]}^{(k)}\}_{P_k \in \tilde{\mathbf{P}}}$ of participants in $\tilde{\mathbf{P}}$.

The adversary $\mathcal{A}$ computes all the parameter changes $\mathbf{Y}_{[b]}$ according to (13). By subtracting the parameter changes of the participants in $\tilde{\mathbf{P}}$, the adversary has

$$\begin{aligned}\mathbf{T}_{[b]} &= \mathbf{Y}_{[b]} - \sum_{P_k \in \tilde{\mathbf{P}}} \mathbf{S}_{[b]}^{(k)} = \mathbf{Y} - \sum_{i=3}^{m} \mathbf{G}_{[b]}^{(i)} \\ &= \mathbf{G}_{[b]}^{(1)} + \mathbf{G}_{[b]}^{(2)},\end{aligned} \tag{25}$$

where the first equality holds according to (10) and the Algorithm 1. In the view of the adversary, $\mathbf{G}_{[b]}^{(2)}$ is randomly drawn from $\mathbb{Z}_p^n$. Hence, the adversary cannot distinguish $\mathbf{T}_{[0]}$ from $\mathbf{T}_{[1]}$. It implies that the adversary cannot distinguish $\mathbf{G}_{[0]}^{(1)}$ from $\mathbf{G}_{[1]}^{(1)}$ with non-negligible probability when $|\hat{\mathbf{P}}| \geq 2$.

Now we analyze the situation that the number of trustful participants $|\hat{\mathbf{P}}| < 2$. Since $1 \leq |\hat{\mathbf{P}}| \leq m$, we assume that the trustful participant $\hat{\mathbf{P}} = \{P_1\}$, and the rest of participants $\tilde{\mathbf{P}} = \{P_2, \cdots, P_m\}$ collude with the cloud server. Similarly, we can also derive that the adversary is capable of obtaining $\mathbf{T}_{[b]} = \mathbf{G}_{[b]}^{(1)}$ by subtracting the parameter changes of participants in $\tilde{\mathbf{P}}$. Here, the adversary can obtain $\mathbf{G}_{[b]}^{(1)}$ in the plaintext. It implies that the adversary can easily distinguish $\mathbf{G}_{[0]}^{(1)}$ from $\mathbf{G}_{[1]}^{(1)}$ when $|\hat{\mathbf{P}}| < 2$. This completes the proof. □

6. Experimental results

In this section, we conduct the experiments to evaluate the performance of our proposed privacy-preserving SSDDL framework. We implement our framework using Tensorflow framework on Ubuntu 16.04 operation system. All the following experiments are performed on a computer with a Intel Core i7 CPU, a GTX 1080 GPU and 64GB RAM.

We evaluate our proposed framework on two widely-used datasets: MNIST [20] and SVHN [25]. MNIST dataset[1] consists of 70000 handwritten digits images of size 28 × 28, among which 60000 images are used for training and the remaining 10000 for testing. SVHN dataset[2] contains 620000 color images of size 32 × 32. Each image is centered around a digit, which is a house number captured from Google's street view images. To be consistent with [34], we use 100000 samples for training and the other 10000 samples for testing. In addition, multi-layer perceptron (MLP) and convolutional neural network (CNN) are adopted. The number of parameters in different models for different datasets are shown in Table 1.

Firstly, we evaluate the performance of our framework on the MNIST dataset, and compare it with the traditional centralized scheme, where the server possesses the entire dataset. In this scenario, the server controls the whole dataset and utilizes the SGD algorithm to train a deep model. Additionally, for comparison purpose, we also re-implement the frameworks DSSGD [34], and HE-DSSGD [29], which is the homomorphic encrypted version of DSSGD. Since the prototype of

[1] http://yann.lecun.com/exdb/mnist/

[2] http://ufldl.stanford.edu/housenumbers

Table 1
The number of model parameters in different models for different datasets.

	MNIST	SVHN
MLP	109, 386	3426250
CNN	3274634	4435146

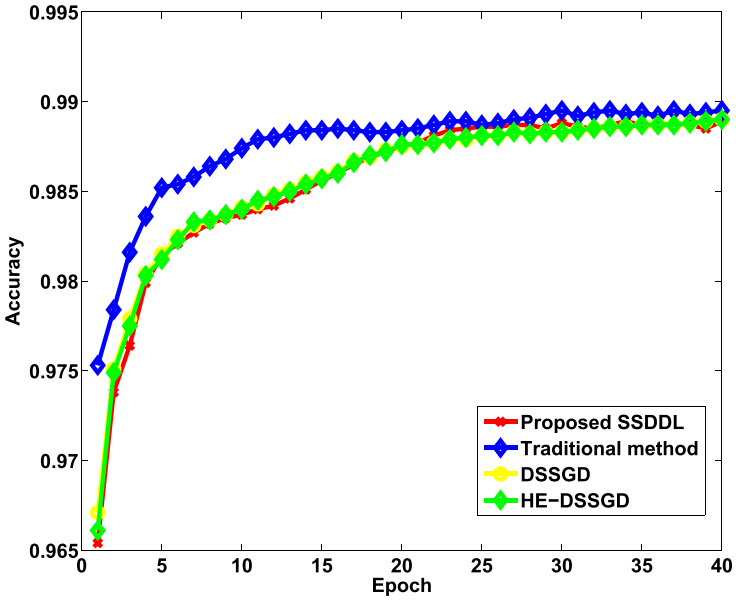


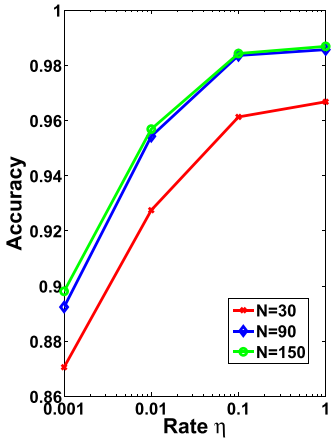
Fig. 5. The comparison of different learning methods.

Bonawitz's protocol [4] focuses on the general machine learning methods (only cryptographic simulations), not the real-world DDL framework, we only provide the theoretical analyses in the following comparison experiments. Here, the number of participants N is fixed to be 90, and the uploading rate η is set to be 0.1. The comparison regarding the accuracy performance is illustrated in Fig. 5. Because Bonawitz's protocol [4], DSSGD [34], HE-DSSGD [29] and our proposed SSDDL are based on distributed SGD methods, it can be inferred that the accuracy performance of these frameworks should be the same. As can be seen in Fig. 5, the accuracy of our proposed SSDDL framework is close to that of DSSGD and HE-DSSGD, which are insecure against participants, as discussed in Section 2. Furthermore, compared with the traditional centralized scheme, our proposed framework achieves almost the same accuracy when the number of epoches is sufficiently large.

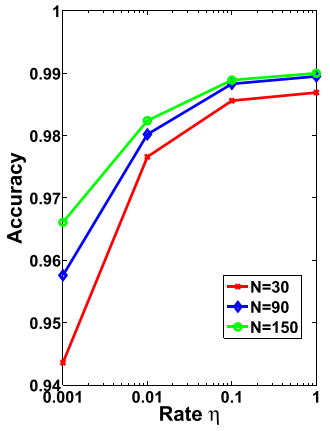
Secondly, we investigate the impact of the different parameter settings on the model accuracy. Specifically, we vary the number of participants N in the range of {30, 90, 150}. Each participant is assigned with 1% of the entire dataset as the local private training data. Also, we vary the uploading rate η in the range of {1, 0.1, 0.01, 0.001}. Then, we examine the mini-batch size M in the range of {1, 10, 20, 30}. The results are shown in Fig. 6. As can be observed, the accuracy of the model grows with respect to the increasing uploading rate η and the decreasing of the mini-batch size M. Additionally, more involving participants usually lead to more accurate model. The reason is that each participant has the same number of training samples, and more participants imply more training samples.

We would also like to investigate the impact of the local epochs n_{epo} on the convergence speed. Here, we fix the parameter $N = 90$ and the uploading rate $\eta = 0.1$. We vary the local epochs n_{epo} in the range {1, 5, 10}. The results are given in Fig. 7. As can be noticed, the accuracy improves with the increasing of the local epochs n_{epo}. It implies that the larger n_{epo} can provide faster convergence rate.

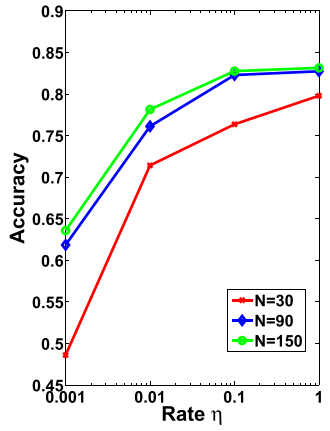
Finally, we show the experimental results on the communication cost of our proposed method, and compare it with the other competing frameworks in Table 2. For the fairness, all the results are conducted in *one round* training among all the participants, which means that the cloud server completes one iteration of training with all the participants. In Table 2, b represents the bit precision and n denotes the number of the parameters. In our proposed framework, the communication cost comes from two modules: SelectGroup and ParaShare. In SelectGroup module, the cloud server broadcasts the global parameters once to all the participants, and hence, the communication cost is nb. In ParaShare module, each participant sends $m - 1$ shares of its parameter change to the other participants, and uploads the aggregation result to the cloud server. Here, since a fraction η of the parameter changes are shared among the participants, the communication cost



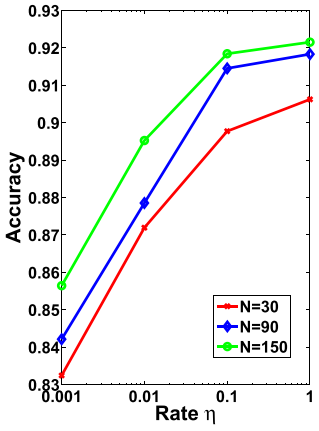
(a) MLP for MNIST, $M = 10$



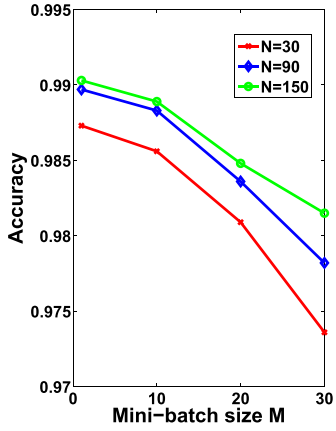
(b) CNN for MNIST, $M = 10$



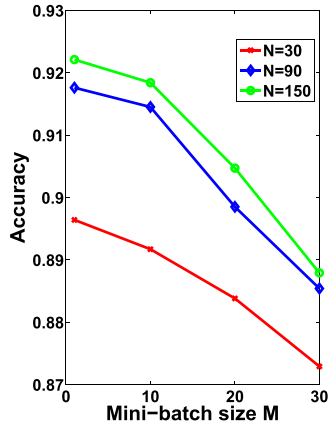
(c) MLP for SVHN, $M = 10$



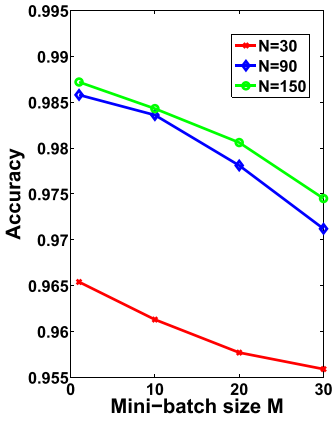
(d) CNN for SVHN, $M = 10$



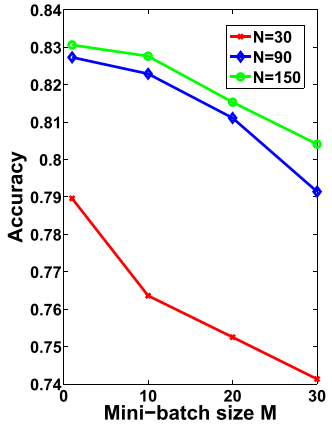
(e) CNN for MNIST, $\eta = 0.1$



(f) CNN for SVHN, $\eta = 0.1$



(g) MLP for MNIST, $\eta = 0.1$



(h) MLP for SVHN, $\eta = 0.1$

Fig. 6. The model accuracy under different parameter settings.

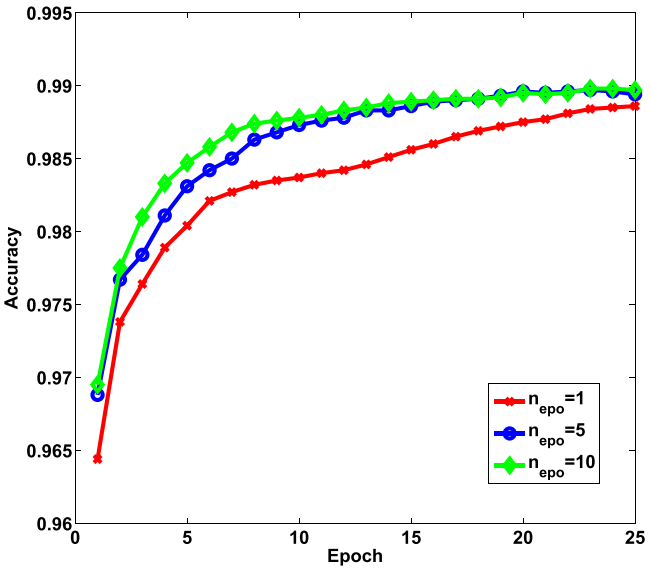


Fig. 7. The convergence rate of the learning model on MNIST with different local epochs n_{epo}.

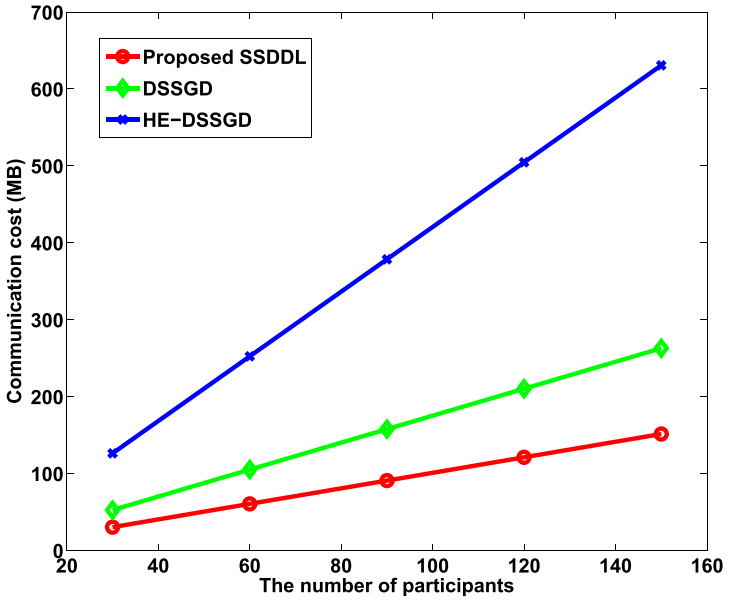


Fig. 8. The communication cost under different number of participants.

can be calculated as $m(m-1)\eta nb + m\eta nb$. Hence, the total cost of our proposed framework in one round of training is $\lceil N/m \rceil (1+m^2\eta)nb$. In DSSGD [34], each participant downloads all the global parameters from the cloud server, and uploads a fraction η of the parameter changes to the cloud server. Thus, the associated communication cost can be calculated as $N(1+\eta)nb$. Compared to DSSGD, the difference in HE-DSSGD framework [29] is that the parameter changes are encrypted via additively HE, leading to the data expansion for the communication. Let ρ be the data expansion factor of the learning with errors (LWE)-based encryption, which is about 2.4 under a typical setting [29]. Here, λ denotes the security parameter in LWE-based encryption. In Bonawitz's protocol [4], each participant sends $2N$ public keys and $5N-4$ secret shares. Here, let b_k and b_s denote the bit length of the public key and the secret shares, respectively. Thus, the associated communication cost can be computed as $N(2Nb_k + 5(N-4)b_s + nb)$, which is of order $O(N^2 + Nn)$. Compared to DSSGD, HE-DSSGD and our proposed SSDDL, which are of order $O(Nn)$, Bonawitz's protocol involves more communication cost to distribute the keys and shares. Regarding the computational cost, as can be seen in Table 2, our proposed SSDDL needs less computational cost than Bonawitz's protocol.

Table 2
The comparison of the communication cost and complexity for different learning frameworks.

Method	Communication cost	Complexity
Proposed SSDDL	$\lceil N/m \rceil (1+m^2\eta)nb$	$O(nN)$
DSSGD [34]	$N(1+\eta)nb$	$O(nN)$
HE-DSSGD [29]	$N(1+\eta)\rho nb$	$O(\lambda nN)$
Bonawitz et al. [4]	$N(2Nb_k+5(N-4)b_s+nb)$	$O(nN^2)$

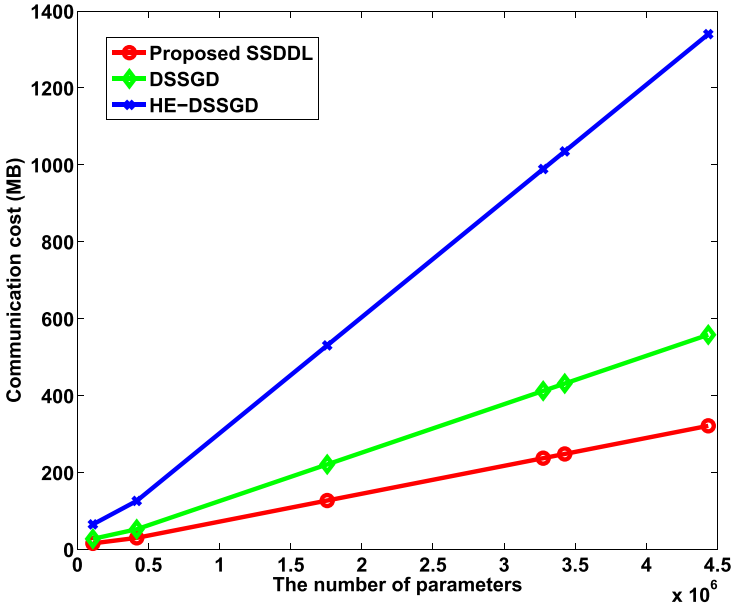


Fig. 9. The communication cost under different number of model parameters.

To measure the impact of different number of participants N on the communication cost, we employ a simple CNN network with 417482 parameters on dataset MNIST and vary N from 30 to 150. Here, the group size $m = 3$ and the rate $\eta = 0.1$. As can be seen in Fig. 8, the communication cost increases almost linearly with respect to the number of participants N, which coincides with our analyses in Table 2.

Prior to concluding this section, let us give some results on the costs of one round with different number of model parameters, which can be shown in Fig. 9. Here, the number of participants $N = 30$, the group size $m = 3$, and the rate $\eta = 0.1$. As can be seen, the communication cost increases almost linearly with respect to the number of parameters n. For instance, when a simple CNN network with 417482 parameters is employed for MNIST dataset, the communication cost of our proposed method in one round is 31 MB, while the costs for DSSGD and HE-DSSGD are 53 MB and 126 MB, respectively. Therefore, our proposed privacy-preserving SSDDL framework not only provides satisfactory privacy protection, but also significantly reduces the communication cost, compared with the state-of-the-art competitors DSSGD [34] and HE-DSSGD [29].

7. Conclusion

We have designed and implemented a privacy-preserving SSDDL framework that enables multiple learning participants to cooperatively train an accurate model with low communication and computational cost. A simple yet effective secret sharing scheme has been adopted to protect the parameter changes. It has been proved theoretically that the local dataset of each participant can be protected in a satisfactory manner against the honest-but-curious cloud server as well as the other participants, even when the cloud server colludes with some participants. Finally, experimental results have been demonstrated to show the superior performance of our proposed privacy-preserving SSDDL framework.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Jia Duan: Writing - original draft, Software, Conceptualization, Methodology. **Jiantao Zhou:** Writing - review & editing, Supervision, Methodology, Formal analysis. **Yuanman Li:** Validation, Data curation, Visualization, Investigation.

Acknowledgments

This work was supported by Macau Science and Technology Development Fund under SKL-IOTSC-2018-2020, 077/2018/A2 and 022/2017/A1, by Research Committee at University of Macau under MYRG2018-00029-FST and MYRG2019-00023-FST, and by Natural Science Foundation of China under 61971476.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H.B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: Proc. ACM Conf. Computer Commun. Security, 2016, pp. 308–318.
[2] M. Barni, C. Orlandi, A. Piva, A privacy-preserving protocol for neural-network-based computation, in: Proc. 8th Workshop Multimedia Security, 2006, pp. 146–151.
[3] M. Bellare, B. Yee, Forward-security in private-key cryptography, in: Proc. RSA Conf. on Cryptographers Track, 2003, pp. 1–18.
[4] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for privacy-preserving machine learning, in: Proc. ACM Conf. Computer Commun. Security, 2017, pp. 1175–1191.
[5] D. Boneh, M. Franklin, Identity-based encryption from the weil pairing, in: Proc. Annual Int. Cryptology Conf., 2001, pp. 213–229.
[6] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, E. Prouff, Privacy-preserving classification on deep neural network, 2017, (Cryptology ePrint Archive, Report 2017/035). http://eprint.iacr.org/2017/035.
[7] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, D. Wong, New algorithms for secure outsourcing of large-scale systems of linear equations, IEEE Trans. Inf. Forensics Secur. 10 (1) (2015) 69–78.
[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder–decoder for statistical machine translation, in: Proc. Empirical Methods Natural Language Process., 2014, pp. 1724–1734.
[9] A. Coates, A. Ng, H. Lee, An analysis of single-layer networks in unsupervised feature learning, in: Proc. Int. Conf. Artificial Intelligence Statistics, 2011, pp. 215–223.
[10] 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: Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1223–1231.
[11] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, J. Makhoul, Fast and robust neural network joint models for statistical machine translation, in: Proc. Conf. Association for Computational Linguistics, 2014, pp. 1370–1380.
[12] J. Duan, J. Zhou, Y. Li, Secure and verifiable outsourcing of large-scale nonnegative matrix factorization (nmf), IEEE Transactions on Services Computing (2019), doi:10.1109/TSC.2019.2911282. 1–1
[13] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (Jul) (2011) 2121–2159.
[14] A. Esteva, B. Kuprel, R.A. Novoa, J. Ko, S.M. Swetter, H.M. Blau, S. Thrun, Dermatologist-level classification of skin cancer with deep neural networks, Nature 542 (7639) (2017) 115.
[15] Z. Fu, F. Huang, K. Ren, J. Weng, C. Wang, Privacy-preserving smart semantic search based on conceptual graphs over encrypted outsourced data, IEEE Trans. Inf. Forensics Secur. 12 (8) (2017) 1874–1884.
[16] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, J. Wernsing, Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy, in: Proc. Int. Conf. Machine Learn., 2016, pp. 201–210.
[17] S. Hu, L.Y. Zhang, Q. Wang, Z. Qin, C. Wang, Towards private and scalable cross-media retrieval, IEEE Transactions on Dependable and Secure Computing (2019). 1–1
[18] P.H. Jin, Q. Yuan, F. Iandola, K. Keutzer, How to scale distributed deep learning? arXiv preprint arXiv:1611.04581 (2016).
[19] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[20] F. Lauer, C.Y. Suen, G. Bloch, A trainable feature extractor for handwritten digit recognition, Pattern Recognit 40 (6) (2007) 1816–1824.
[21] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A.Y. Ng, On optimization methods for deep learning, in: Proc. Int. Conf. Machine Learn., 2011, pp. 265–272.
[22] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, B.-Y. Su, Scaling distributed machine learning with the parameter server, in: Proc. Conf. Operating Syst. Design Implementation, 2014, pp. 583–598.
[23] J. Liu, M. Juuti, Y. Lu, N. Asokan, Oblivious neural network predictions via minionn transformations, in: Proc. ACM Conf. Computer Commun. Security, 2017, pp. 619–631.
[24] X. Ma, F. Zhang, X. Chen, J. Shen, Privacy preserving multi-party computation delegation for deep learning in cloud computing, Inf Sci (Ny) 459 (2018) 103–116.
[25] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning, in: Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn., 2011, pp. 1–9.
[26] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, et al., Deepid-net: Deformable deep convolutional neural networks for object detection, in: Proc. IEEE Conf. Computer Vision Pattern Recognit., 2015, pp. 2403–2412.
[27] N. Phan, Y. Wang, X. Wu, D. Dou, Differential privacy preservation for deep auto-encoders: an application of human behavior prediction, in: Proc. 30th AAAI Conf. Artificial Intelligence, 2016, pp. 1309–1316.
[28] L.T. Phong, Privacy-preserving stochastic gradient descent with multiple distributed trainers, in: Proc. Int. Conf. Network Syst. Security, 2017, pp. 510–518.
[29] L.T. Phong, Y. Aono, T. Hayashi, L. Wang, S. Moriai, Privacy-preserving deep learning via additively homomorphic encryption, IEEE Trans. Inf. Forensics Secur. 13 (5) (2018) 1333–1345.
[30] L.T. Phong, T.T. Phuong, Privacy-preserving deep learning via weight transmission, IEEE Transactions on Information Forensics and Security (2019), doi:10.1109/TIFS.2019.2911169. 1–1
[31] R. Raina, A. Madhavan, A.Y. Ng, Large-scale deep unsupervised learning using graphics processors, in: Proc. Int. Conf. Machine Learn., 2009, pp. 873–880.
[32] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: A unified embedding for face recognition and clustering, in: Proc. IEEE Conf. Computer Vision Pattern Recognit., 2015, pp. 815–823.
[33] A. Shamir, How to share a secret, Commun ACM 22 (11) (1979) 612–613.
[34] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: Proc. ACM Conf. Computer Commun. Security, 2015, pp. 1310–1321.
[35] M. Steiner, G. Tsudik, M. Waidner, Diffie-hellman key distribution extended to group communication, in: Proc. ACM Conf. Computer Commun. Security, 1996, pp. 31–37.

[36] C. Szegedy, A. Toshev, D. Erhan, Deep neural networks for object detection, in: Proc. Adv. Neural Inf. Process. Syst., 2013, pp. 2553–2561.
[37] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: Closing the gap to human-level performance in face verification, in: Proc. IEEE Conf. Computer Vision Pattern Recognit., 2014, pp. 1701–1708.
[38] M. Van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully homomorphic encryption over the integers, in: Proc. Annual Int. Conf. Theory Appl. Cryptographic Techniques, 2010, pp. 24–43.
[39] C. Wang, N. Cao, K. Ren, W. Lou, Enabling secure and efficient ranked keyword search over outsourced cloud data, IEEE Trans. Parallel Distrib. Syst. 23 (8) (2012) 1467–1479.
[40] C. Wang, K. Ren, J. Wang, Secure and practical outsourcing of linear programming in cloud computing, in: Proc. IEEE Int. Conf. Computer Commun., 2011, pp. 820–828.
[41] C. Wang, K. Ren, J. Wang, Secure optimization computation outsourcing in cloud computing: a case study of linear programming, IEEE Trans. Comput. 65 (1) (2016) 216–229.
[42] C. Wang, K. Ren, J. Wang, K.M.R. Urs, Harnessing the cloud for securely solving large-scale systems of linear equations, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1172–1181.
[43] Q. Wang, S. Hu, M. Du, J. Wang, K. Ren, Learning privately: Privacy-preserving canonical correlation analysis for cross-media retrieval, in: IEEE Conf. Computer Commun., 2017, pp. 1–9.
[44] Z. Xu, C. Wang, K. Ren, L. Wang, B. Zhang, Proof-carrying cloud computation: the case of convex optimization, IEEE Trans. Inf. Forensics Secur. 9 (11) (2014) 1790–1803.
[45] J. Yuan, S. Yu, Privacy preserving back-propagation neural network learning made practical with cloud computing, IEEE Trans. Parallel Distrib. Syst. 25 (1) (2014) 212–221.
[46] L.Y. Zhang, Y. Zheng, J. Weng, C. Wang, Z. Shan, K. Ren, You can access but you cannot leak: defending against illegal content redistribution in encrypted cloud media center, IEEE Transactions on Dependable and Secure Computing (2018). 1–1
[47] Q. Zhang, L.T. Yang, Z. Chen, Privacy preserving deep computation model on cloud for big data feature learning, IEEE Trans. Comput. 65 (5) (2016) 1351–1362.
[48] T. Zhang, Q. Zhu, Dynamic differential privacy for admm-based distributed classification learning, IEEE Trans. Inf. Forensics Secur. 12 (1) (2017) 172–187.