Information Sciences 603 (2022) 190–209

Contents lists available at ScienceDirect

Information Sciences

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

CORK: A privacy-preserving and lossless federated learning scheme for deep neural network

Jiaqi Zhao [a], Hui Zhu [a,*], Fengwei Wang [a], Rongxing Lu [b], Hui Li [a], Jingwei Tu [c], Jie Shen [c]

[a] State Key Laboratory of Integrated Networks Services, Xidian University, China
[b] Faculty of Computer Science, University of New Brunswick, Canada
[c] China Mobile (Suzhou) Software Technology Co., Ltd, China

ARTICLE INFO

Article history:
Received 8 May 2021
Received in revised form 15 January 2022
Accepted 24 April 2022
Available online 28 April 2022

Keywords:
Federated learning
Deep neural network
Privacy-preserving
Secure aggregation
Model perturbation

ABSTRACT

With the advance of machine learning technology and especially the explosive growth of big data, federated learning, which allows multiple participants to jointly train a high-quality global machine learning model, has gained extensive attention. However, in federated learning, it has been proved that inference attacks could reveal sensitive information from both local updates and global model parameters, which threatens user privacy greatly. Aiming at the challenge, in this paper, a privacy-preserving and lossless federated learning scheme, named CORK, is proposed for deep neural network. With CORK, multiple participants can train a global model securely and accurately with the assistance of an aggregation server. Specifically, we first design a drop-tolerant secure aggregation algorithm FTSA, which ensures the confidentiality of local updates. Then, a lossless model perturbation mechanism PTSP is proposed to protect sensitive data in global model parameters. Furthermore, the neuron pruning operation in PTSP can reduce the scale of models, which thus improves the computation and communication efficiency significantly. Detailed security analysis shows that CORK can resist inference attacks on both local updates and global model parameters. In addition, CORK is implemented with real MNIST and CIFAR-10 datasets, and the experimental results demonstrate that CORK is indeed effective and efficient.

1. Introduction

In recent years, the deep neural network (DNN) model has been widely applied and gained huge success in many fields (e.g., natural language process [1], computer vision [2], human–machine game [3], and so on.), bringing great convenience to people's life. Meanwhile, due to explosive growth of data volume generated by distributed devices, coupled with privacy concerns of data collection, the concept of federated learning has been introduced by Google [4], which essentially involves training a high-quality global model over multiple participants while remaining data localized, as shown in Fig. 1. In particular, during each training round of federated learning, participants first perform a stochastic gradient descent (SGD) algorithm locally on their training data to generate the local updates. After that, the local updates are further aggregated by aggregation server to update the global model parameters.

Nevertheless, there are still many privacy issues in federated learning. Many previous research works have demonstrated that the inference attacks could reveal sensitive information of participants from both local updates and global model parameters [5,6]. On the one hand, local updates are derived from training data of participants, thus they contain massive

* Corresponding author.
E-mail address: zhuhui@xidian.edu.cn (H. Zhu).

https://doi.org/10.1016/j.ins.2022.04.052
0020-0255/

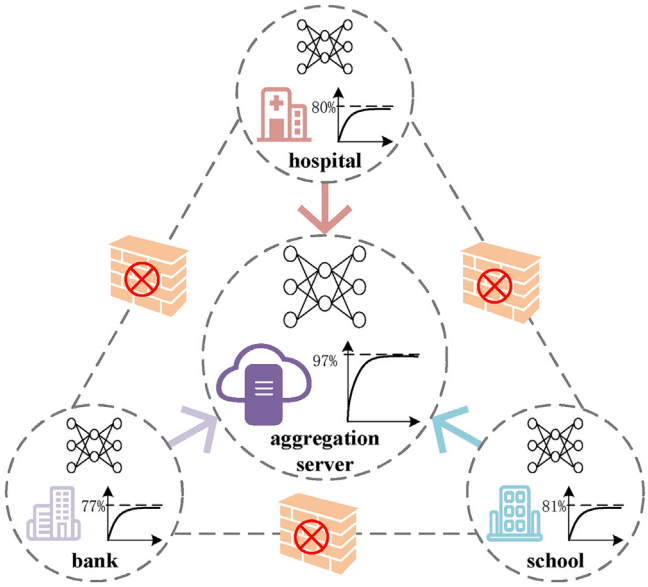


Fig. 1. The architecture of federated learning.

sensitive information. Only with analyzing model updates in just a few rounds, while without any auxiliary knowledge, the aggregation server could infer a certain participant's data information (i.e., membership, class representation, or property), or even reconstruct its raw local training data [7–9]. On the other hand, since the DNN model appears to internally mine useful information in training data, the global model parameters also include sensitive information. For example, by calculating the difference between consecutive global model parameters, an honest-but-curious participant can get the aggregated local updates of other participants, then infer the membership and properties of their data in a certain training round [10]. Therefore, there is an urgent need to design privacy-preserving federated learning schemes for the DNN model.

In order to solve the above privacy issues, plenty of privacy-preserving schemes have been proposed, which can mainly be classified into two categories: based on secure aggregation (SA) or differential privacy (DP). Specifically, with an SA algorithm, an aggregation server can only obtain the summation of multiple participants' local updates, while it is not able to see any concrete local update of a solo participant [11–16]. As a result, SA-based schemes can well protect the sensitive data in local updates from the aggregation server, but it's hard to prevent inference attacks on the global model parameters. In addition to the SA algorithm, DP is a common method to protect data privacy in federated learning. In the popular differential privacy schemes, the local updates are perturbed randomly by participants at each round [17–20]. DP-based schemes can provide strong privacy guarantees, but it inevitably reduces the accuracy of the trained model.

In this paper, we propose a privacy-preserving and lossless federated learning scheme for DNN, named CORK. With CORK, multiple participants can collaborate to train a DNN model securely and accurately with the assistance of an aggregation server. By combining our proposed drop-tolerant secure aggregation algorithm FTSA and lossless model perturbation mechanism PTSP, sensitive data in both local updates and global model parameters are protected well during training. Furthermore, the neuron pruning operation in PTSP can reduce the scale of models, which improves the computation and communication efficiency significantly. Specifically, our contributions are the following:

- CORK protects sensitive data in *both local updates and global model parameters*. During training, the local updates are encrypted by FTSA, which can keep confidential to the aggregation server. Besides, PTSP changes the orders and values of global model parameters greatly, which makes an honest-but-curious participant impossible to infer others' sensitive data by comparing the consecutive global model parameters. Therefore, sensitive data are protected well from the inference attacks of aggregation server and other participants in CORK.
- CORK achieves *lossless and drop-tolerant* federated learning for DNN. In federated learning, a participant may drop out midway due to connectivity or power constraints. In this regard, through applying the Shamir secret sharing in FTSA, even if some participants drop out at a training round, it could still aggregate the local updates of remaining participants. Moreover, in CORK, PTSP prunes and merges redundant neurons in the DNN model, which doesn't cause a loss of model accuracy.
- CORK is efficient in both computation and communication overhead. At each training round, the computation and communication overhead is significantly reduced with the neuron pruning operation in PTSP. In addition, the evaluation on real MNIST and CIFAR-10 datasets shows that our scheme is effective and efficient, which can be implemented in a real environment.

The rest of this paper is organized as follows. In Section 2, we define the models and propose our design goal. In Section 3, we outline some building blocks in CORK. In Section 4, we introduce our proposed CORK detailedly, followed by the security analysis and performance evaluation in Section 5 and Section 6. Finally, we review the related works in Section 7 and draw a conclusion in Section 8.

2. Models, Security Requirements and Design Goal

In this section, we first outline the system model, threat model, and security requirements in our scenario. After that, we identify our design goal.

2.1. System Model

In our system model, we mainly focus on how multiple participants collaboratively train a global DNN model securely and accurately. Each participant is equipped with a computer or workstation, which can store their training data locally and connect with the aggregation server. Specifically, the system consists of three parts: (1) trusted authority (TA); (2) aggregation server (AS); (3) participants, as shown in Fig. 2.

- TA is a trusted authority, which is responsible for initializing the system by generating public parameters and distributing secret keys for the aggregation server and corresponding participants. Afterward, TA will keep offline.
- AS is the aggregation server with sufficient computing resources but no training data, which works as a collaborator to help participants train the DNN model. At each training round, AS is responsible for perturbing the global model parameters and aggregating the encrypted local updates from multiple participants.
- Participants represented as $\{P_1, \ldots, P_n\}$ are some institutions (e.g., school, bank, hospital, and so on) with their local training data. At each training round, each $P_i \in \{P_1, \ldots, P_n\}$ generates its local update through executing an SGD algorithm on its local training data, then send the encrypted local update to AS for updating the global model parameters.

2.2. Threat Model and Security Requirements

In our threat model, we consider that AS and participants are honest-but-curious (semi-honest). Specifically, at each round of training, AS honestly performs the perturbation and aggregation operations, but attempts to infer the training data of participants by analyzing the received local updates. Similarly, each participant executes the training and encryption processes honestly but tries to infer other participants' sensitive data from the global model parameters, which will be introduced detailedly in Section 3.3. Besides, for practical considerations, AS could collude with one or more participants to infer other participants' sensitive information. It is noteworthy that there are still many other types of attacks (e.g., poison attack, evasion attack, etc.) in federated learning. Since our CORK focuses on protecting the sensitive data of participants during the model training process, these attacks are currently out of the scope of this paper and will be considered in future work. Under the above threat model, CORK needs to satisfy the following security requirements.

- *Ensuring the privacy of data in local updates.* Generally, the local updates are derived by performing an SGD algorithm on a participant's local data. Once the local update of any participant is made public, its sensitive information will be leaked, which may cause serious privacy problems. Therefore, during the training process, the local updates of participants should be protected.
- *Ensuring the privacy of data in global model parameters.* If only the local updates are protected, through analyzing the global model parameters, an honest-but-curious participant is still able to infer other participants' sensitive information in every training round. Therefore, we should also guarantee the privacy of global model parameters.

2.3. Design Goal

Under the system model and security requirements mentioned above, in order to achieve the goal of privacy-preserving and lossless federated learning for DNN, the following three objectives should be satisfied.

- *Ensure privacy-preservation during federated learning.* Privacy is often a significant concern in federated learning. Once the sensitive data are disclosed, it may lead to serious consequences. Therefore, CORK should protect participants' sensitive data in both local updates and global model parameters against inference attacks.
- *Achieve lossless and drop-tolerant federated learning for DNN.* In federated learning, training data are distributively stored in multiple institutions, which may not be independent and identically distributed. Therefore, in order to obtain a high-quality DNN model, the correctness of local training and model aggregation should be first guaranteed. Then, we should ensure that the neuron pruning and shuffling operations of PTSP will not affect the accuracy of models. Moreover, due to the connectivity or power constraints of participants in federated learning, CORK should be tolerant of participants dropping out.

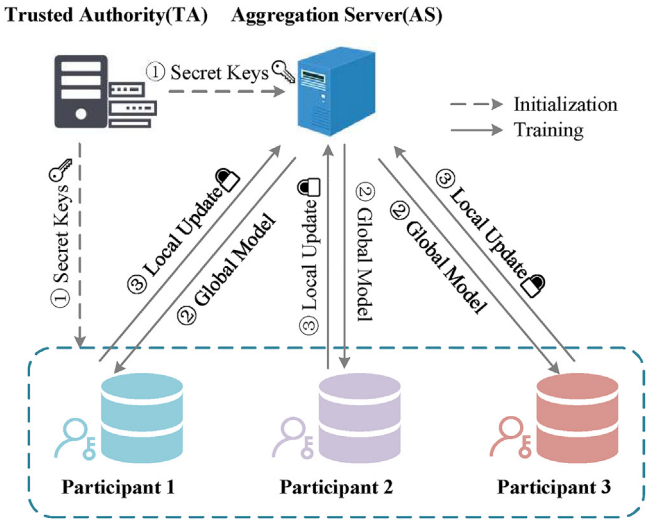


Fig. 2. System model under consideration.

- *Low computation and communication overhead.* Although the computational capabilities of computers and workstations are increasing rapidly, it is still difficult for participants to handle large-scale DNN model training. Meanwhile, since federated learning requires multi-round and long-term interaction, communication overhead and stability are still the bottlenecks that affect the training efficiency. Considering the above factors, the proposed CORK should accomplish low overhead in both computation and communication.

3. Preliminaries

In this section, we review some preliminaries related to our scheme.

3.1. Deep Neural Network and Neuron Pruning

The deep neural network [21,22] is a series of algorithms that aims to learn the characteristics of training data, which can be used to achieve regression, clustering, classification, and prediction in many fields.

As shown in Fig. 3, a DNN model consists of an input layer, several hidden layers, and an output layer. The connection between two neurons can be represented by a float number called weight, and the connection between two layers is a weight matrix written as θ.

In our scheme, we focus on the supervised training setting, which means that a piece of training data contains a feature vector x and a label y. For example, given the training data $x = \{x_1, x_2, x_3, x_4\}$ as input n_0, the final output $n_3 = \{o_1, o_2\}$ is obtained by repeatedly calculating the following formula ($k = 1, 2, 3$):

$$n_k = f(n_{k-1}\theta_k + b_k),$$

where f is the activation function (e.g., Sigmoid, ReLU, etc.), b_k is the bias unit, and n_k is the output of the kth layer's neurons. After getting the final output, the loss function $\mathcal{L}$ is calculated as

$$\mathcal{L}(x, y, \theta) = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2.$$

In order to find suitable weights that minimize the loss function $\mathcal{L}$, the mini-batch SGD (MB-SGD) algorithm is used, which updates the weights iteratively. At each iteration, the weights are updated as follows:

$$\theta \leftarrow \theta - \alpha \cdot \frac{1}{|B|}\sum_{x \in B} \nabla_\theta \mathcal{L}(x, y, \theta),$$

where B is a batch of training data chosen randomly, α is the learning rate, and $\nabla_\theta \mathcal{L}(x, y, \theta)$ is the derivatives of the loss function to the weights. Finally, the above iterations terminate until the loss function $\mathcal{L}$ converges or until the maximum iteration is reached.

In many practical applications, a DNN model usually contains millions of weights. To reduce the network complexity and solve the over-fitting problem, neuron pruning [23] is proposed. The main method of neuron pruning is to prune redundant, non-informative weights in a DNN model. Especially, Srinivas and Babu [24] propose a method to prune neurons without knowing the distribution of data, which can ensure that the network's overall structure keeps the same. We apply their method as a building block of PTSP to perturb the global model parameters.

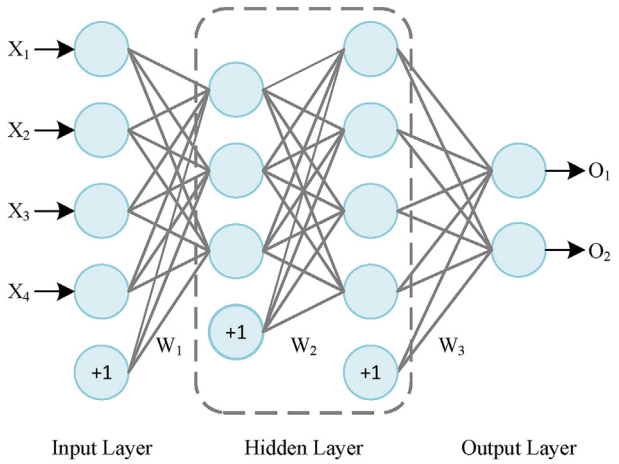


Fig. 3. A DNN model composed of an input layer, two hidden layers and an output layer.

3.2. Federated Averaging

Federated learning [4] is a distributed framework where multiple participants (e.g., different institutions) jointly train a high-quality machine learning model while keeping their data localized. In this framework, the global model parameters are maintained by the aggregation server. At each training round, every active participant independently computes a local update based on its training data, then sends it to the aggregation server for updating the global model parameters.

In our scheme, we adopt an advanced federated learning algorithm, namely Federated Averaging [25], which is a practical federated learning algorithm proposed for DNN, as shown in Algorithm 1. Specifically, by combining model averaging and SGD algorithms, federated averaging is suitable for training data that is not independently and identically distributed, and it can greatly reduce the communication rounds.

Algorithm 1: Federated Averaging

Parameters: number of clients K, batch size B, local training epoch E, learning rate α, client fraction C.

procedure SERVER
  initialize w_0.
  for each round $t = 1, 2, \ldots$ **do**
    $m \leftarrow max(C \cdot K, 1)$.
    $S_t \leftarrow$ random set of m clients.
    for each client $k \in S_t$ **do**
      $w_{t+1}^k \leftarrow \text{CLIENT}(k, w_t)$.
    end for
      $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$.
end for
end procedure
procedure CLIENT(k, w)
  $\mathcal{B} \leftarrow$ split D_k into batches of size B.
  for each local epoch i from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \alpha \nabla \mathcal{L}(w; b)$.
    end for
  end for
  return w to server.
end procedure

3.3. Inference Attacks in Federated Learning

Although there is no raw data sharing in federated learning, which makes progress to protect participants' local training data, the inference attacks could still exploit sensitive information of participants on local updates and global model parameters.

Since local updates are the inner products or convolutions of the errors and the data features, they could be used to infer massive sensitive information of local data, such as class representatives, membership, and properties, even to recover the

original training data. For example, the non-zero local updates of the embedding layer in natural language processing model could reveal which words appear in the training data [10]. Besides, a method called Deep Leakage of Gradient (DLG) could obtain both the features and labels of training data by observing the local updates in just a few iterations [8,9].

Besides, since DNN appears to learn many internal features of training data that are not apparently related to the main tasks, global model parameters could also leak extra information about the unintended features of participants' training data. For example, when training a binary gender classifier, an honest-but-curious participant can save the snapshot of the global model parameters, then infer when a specific person first appears in the photos by exploiting the difference between the consecutive snapshots [10].

3.4. Paillier Cryptosystem

Paillier cryptosystem [26] is a novel probabilistic encryption based on the composite residuosity problem. In our scheme, we take advantage of Paillier cryptosystem's additive homomorphic property to construct our FTSA algorithm. Here, we briefly review the Paillier cryptosystem:

- **Key Generation:** Select two large primes $|p| = |q| = \kappa$, compute $N = p \cdot q$ and $\lambda = lcm(p-1, q-1)$. Then, select a random integer $g \in \mathbb{Z}^*_{N^2}$ satisfying $gcd(L(g^\lambda \bmod N^2), N) = 1$, where $L(u) = (u-1)/N$ for $u \in \mathbb{Z}^*_{N^2}$. Finally, the public key is (N, g) and the corresponding secret key is λ.
- **Encryption:** Given a message m, random number $r \in \mathbb{Z}^*_{N^2}$ is chosen and the ciphertext can be computed as $c = E_{pk}(m) = g^m r^N \bmod N^2$.
- **Decryption:** Given a ciphertext $c < N^2$, the plaintext m can be retrieved as $m = D_{sk}(c) = \frac{L(c^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \bmod N^2$.
- **Homomorphic Property:** Given two ciphertexts $E_{pk}(m_1)$ and $E_{pk}(m_2)$, we have $E_{pk}(m_1)E_{pk}(m_2) = g^{m_1+m_2}(r_1 r_2)^N \bmod N^2 = E_{pk}(m_1 + m_2)$.

4. Proposed Scheme

In this section, we present a detailed description of our CORK scheme, which mainly consists of four phases: (1) System initialization; (2) Model perturbation and distribution; (3) Local training and encryption; (4) Secure aggregation and model recovery. The overview of CORK is shown in Fig. 4. At first, TA generates and distributes public parameters and keys. Then, n participants train the model iteratively with the assistance of AS. At each training round, AS perturbs and distributes the global model parameters to participants. Then every participant independently calculates the local update on its local training data and sends the encrypted local update to AS. After that, the local updates of multiple participants are aggregated as the aggregation update, which is recovered by AS to obtain the new global model parameters. Finally, the above iterations terminate until the global model converges or until the maximum round is reached.

For describing more clearly, we first introduce the main idea of CORK. Then, the drop-tolerant secure aggregation algorithm FTSA and the lossless model perturbation mechanism PTSP are proposed, which are two building blocks of CORK. Finally, we describe the four phases of CORK detailedly, followed by the correctness analysis. In Table 1, we list the common notations used in CORK.

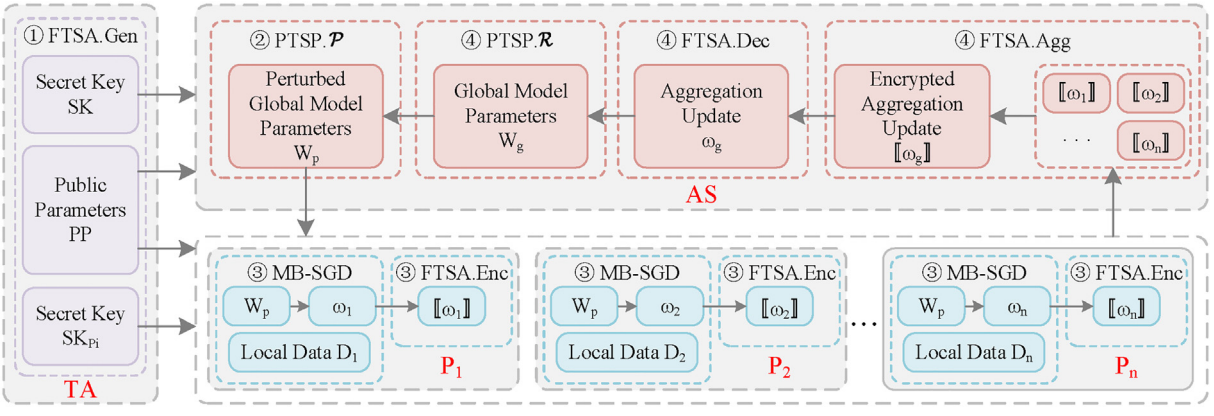


Fig. 4. Overview of CORK.

4.1. Main Idea of CORK

In traditional federated learning schemes, the sensitive data could be inferred not only from local updates but also from global model parameters, as mentioned in Section 3.3.

In CORK, by modifying the Paillier cryptosystem, we propose the drop-tolerant secure aggregation algorithm FTSA. It achieves that AS cannot see the local update of a single participant but can still obtain the exact aggregation of multiple local updates at each training round. Therefore, the sensitive data in the local updates are protected well against the honest-but-curious AS.

In addition, to ensure the privacy of sensitive data in global model parameters, we design the lossless model perturbation mechanism PTSP, which can change the orders and values of global model parameters greatly. With PTSP, it is impossible for a participant to match the neuron positions of consecutive global models, and further obtain the aggregation of other participants' local updates. Therefore, CORK protects the sensitive data in the global model parameters, and a participant cannot infer the sensitive data used in a certain training round.

4.2. Drop-tolerant Secure Aggregation Algorithm FTSA

In this section, we propose the FTSA algorithm detailedly, which is based on Paillier cryptosystem. And it consists of four functions, namely $Gen(\cdot), Enc(\cdot), Agg(\cdot)$, and $Dec(\cdot)$.

- $FTSA.Gen(\kappa, t, \mathcal{A}_p) \rightarrow (PP, SK, \{SK_{P_i}\}_{i\in\mathcal{A}_p})$: At first, given a secure parameter κ, TA generates parameters of the Paillier cryptosystem, which consist of the secret key λ and the public key (g, N). Next, TA chooses a big prime p' and compute $h = g^{p'} \bmod N^2$. Finally, given a participant list $\mathcal{A}_p$ and a threshold t, TA announces the public parameters $PP =< \kappa, t, S, g, h, N >$, where S is the size of list $\mathcal{A}_p$.
  For AS, TA sends the secret key $SK =< \lambda, p' >$. For participants, TA first chooses $t-1$ random integers $a_1, a_2, \ldots, a_{t-1} \in \mathbb{Z}_p$, and constructs a polynomial
  $$f(x) = a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{t-1} \cdot x^{t-1} \bmod p. \tag{1}$$
  Then, with a random number $s \in \mathbb{Z}_N$, for each $P_i \in \mathcal{A}_p$, TA sends its secret key $SK_{P_i} = s^{q \cdot f(i)} \bmod N^2$.
- $FTSA.Enc(PP, SK_{P_i}, \omega_i) \rightarrow [\![\omega_i]\!]$: Given a message ω_i belonging to $\mathbb{Z}_N$, the ciphertext $[\![\omega_i]\!]$ can be computed as:
  $$[\![\omega_i]\!] = g^{\omega_i} \cdot h^{r_i} \cdot SK_{P_i}^{(S-1)! \prod_{j\in\mathcal{A}_f, j\neq i} \frac{j}{j-i}} \bmod N^2, \tag{2}$$
  where r_i is a random integer that satisfies $r_i \in \mathbb{Z}_N$ and $|r_i| < \frac{\kappa}{2}$ and $\mathcal{A}_f$ is the list of participants who are still active to join in the following aggregation process. Moreover, $(S-1)!$ is a pre-calculated constant value to ensure that the exponent of SK_{P_i} is an integer.
- $FTSA.Agg(PP, \{[\![\omega_i]\!]\}_{i\in\mathcal{A}_f}) \rightarrow [\![\omega_g]\!]$: After receiving $\{[\![\omega_i]\!]\}_{i\in\mathcal{A}_f}$, AS can aggregate the ciphertexts as:
  $$[\![\omega_g]\!] = \prod_{i\in\mathcal{A}_f} [\![\omega_i]\!] \bmod N^2. \tag{3}$$
  - $FTSA.Dec(PP, SK, [\![\omega_g]\!]) \rightarrow \omega_g$: Given the aggregation update $[\![\omega_g]\!]$, AS can decrypt it through computing:
    $$\omega_g = \sum_{i\in\mathcal{A}_f} \omega_i = \frac{L([\![\omega_g]\!]^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \bmod N \bmod p'.$$

4.3. Model Perturbation Mechanism PTSP

In this section, we propose the PTSP mechanism, which consists of the perturbation function $\mathcal{P}(\cdot)$ and recovery function $\mathcal{R}(\cdot)$.

4.3.1. Perturbation Function

As shown in Algorithm 2, function $\mathcal{P}(\cdot)$ can perturb the model parameters through two steps, namely, neuron pruning and neuron shuffling. The input N_p and N_s represent the numbers of neuron pruning and shuffling, which decide the degree of model perturbation.

- Neuron pruning: For the lth layer, the neuron pruning operation computes the distances between all neuron pairs to form a distance matrix ξ, then prune $N_p^{(l)}$ neurons iteratively. For each pruning, it first finds the minimum value in ξ, represented as $min(\xi)$. Then, the corresponding index of $min(\xi)$ is written as (m, n), which shows that the mth and nth neurons

are the most similar in the lth layer. After that, it removes the nth neuron and merges it to the mth neuron via deleting the nth column in W_l, updating the mth row in W_{l+1} as $W_{l+1}^{(m,\cdot)} = W_{l+1}^{(m,\cdot)} + W_{l+1}^{(n,\cdot)}$, and deleting the nth row in W_{l+1}. Finally, the matrix ξ is updated for the next pruning by removing the mth column and row and recalculating the nth column.

- Neuron shuffling: After finishing $N_p^{(l)}$ prunings, the neurons in the lth layer is further perturbed by neuron shuffling operation, which shuffles the neurons for $N_s^{(l)}$ times. Specifically, every shuffling exchanges the pth and qth neurons randomly by performing $W_i^{(\cdot,p)} \leftrightarrow W_i^{(\cdot,q)}$ and $W_{i+1}^{(p,\cdot)} \leftrightarrow W_{i+1}^{(q,\cdot)}$, then adds the random pair (p,q) to record $R_s^{(l)}$ for recovery. Finally, function $\mathcal{P}(\cdot)$ returns the perturbed model W_p and shuffling records R_s.

In order to describe our perturbation function $\mathcal{P}(\cdot)$ more clearly, the changes in the DNN model and weight matrix can be observed respectively in Fig. 5, where the solid line, dotted line, and red shapes mean remaining, removed, and merged neurons and weights respectively, which shows the orders and values of model parameters are changed greatly by function $\mathcal{P}(\cdot)$.

4.3.2. Recovery Function

Similarly, given perturbed W_p, function $\mathcal{R}(\cdot)$ can recover it to the complete model W_g. For the lth layer, $\mathcal{R}(\cdot)$ firstly recovers the orders of neurons based on record $R_s^{(l)}$. Then, according to the numbers of pruned neurons $N_p^{(l)}$, function $\mathcal{R}(\cdot)$ fills W_l and W_{l+1} with random vectors, where $columns(W_l)$ denotes the total number of columns in matrix W_l. Finally, the recovered model parameter W_g is returned.

Algorithm 2: Model Perturbation Mechanism PTSP

Parameters: model parameter $W_g, W_p = \{W_0, W_1, \ldots, W_h\}$, pruning number $N_p = \{N_p^{(1)}, \ldots, N_p^{(h)}\}$, shuffling number $N_s = \{N_s^{(1)}, \ldots, N_s^{(h)}\}$, shuffling record $R_s = \{R_s^{(1)}, \ldots, R_s^{(h)}\}$.

function $\mathcal{P}(W_g, N_p, N_s)$
  for $l = 1, 2, \ldots, h$ **do**
    Compute the distance matrix ξ in the lth layer.
    for $p = 1, 2, \ldots, N_p^{(l)}$ **do** ▷neuron pruning
      $(m, n) \leftarrow index(min(\xi))$.
      Delete $W_l^{(\cdot,n)}$.
      $W_{l+1}^{(m,\cdot)} = W_{l+1}^{(m,\cdot)} + W_{l+1}^{(n,\cdot)}$.
      Delete $W_{l+1}^{(n,\cdot)}$.
      Update ξ.
    end for
    for $s = 1, 2, \ldots, N_s^{(l)}$ **do** ▷neuron shuffling
      Generate a random pair (i, j), and add to $R_s^{(l)}$.
      $W_l^{(\cdot,i)} \leftrightarrow W_l^{(\cdot,j)}$, $W_{l+1}^{(i,\cdot)} \leftrightarrow W_{l+1}^{(j,\cdot)}$.
    end for
  end for
  $W_p \leftarrow W_g$.
  return W_p, R_s
end function
function $\mathcal{R}(W_p, R_s, N_p)$
  for $l = 1, 2, \ldots, h$ **do**
    for $(p, q) \in R_s^{(l)}$ **do**
      $W_l^{(\cdot,p)} \leftrightarrow W_l^{(\cdot,q)}$, $W_{l+1}^{(p,\cdot)} \leftrightarrow W_{l+1}^{(q,\cdot)}$.
  end for
    for $f = 1, 2, \ldots, N_p^{(l)}$ **do**
      Choose a random integer $n < columns(W_l)$.
      Insert random vectors into $W_l^{(\cdot,n)}$ and $W_{l+1}^{(n,\cdot)}$.
    end for
  end for
  $W_g \leftarrow W_p$.
  return W_g
end function

Table 1
Notations of CORK.

Notations	Definition
$\mathcal{A}_p$	The list of active participants.
κ	Security parameter.
t	Threshold.
PP	Public parameters.
SK, SK_{P_i}	Secret keys of AS and P_i.
W_g	Global model parameters.
$\mathcal{A}_f$	The list of participants who have completed training.
ω_i	The local update of P_i.
ω_g	Aggregation update.
$[\![\omega_i]\!]$	The encrypted local update of P_i.
$[\![\omega_g]\!]$	Encrypted aggregation update.
N_p, N_s	Pruning number and shuffling number.
R_s	Shuffling record.
$W_l^{(i,\cdot)}, W_l^{(\cdot,j)}$	The ith row and jth column of W_l.
W_p	Perturbed global model parameters.
D_i	The local training data of P_i.
γ_i	The encryption parameter of P_i.
N_d, N_i	Total number of data, number of P_i's local data.
α, B, E	Learning rate, batch size and local training epoch.

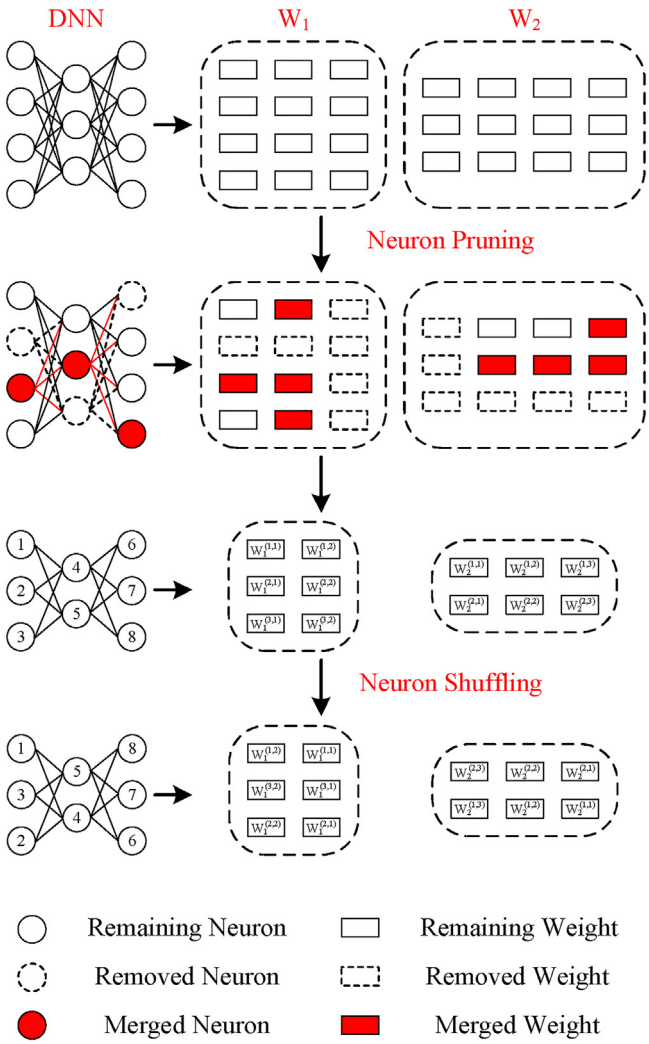


Fig. 5. Model perturbation function of PTSP.

4.4. Description of CORK

In this section, we describe the four phases of CORK. In order to describe more clearly, we show the protocol process in Algorithm 3.

4.4.1. System Initialization

In this phase, TA first generates and distributes public parameters and keys to AS and corresponding P_i. Then, AS initializes the parameters in model training.

- **Step 1. System Parameter Initialization**
  At first, TA generates the participant list $\mathcal{A}_p = \{P_1, P_2, \cdots, P_n\}$, and the number of P_i' local data is denoted as N_i.
  Then, TA selects secure parameter κ and threshold $t(t < n)$, where t is the minimum number of participants to complete model training.
  Finally, TA executes function $FTSA.Gen(\kappa, t, \mathcal{A}_p)$ to compute public parameters PP and generates secret keys SK and SK_{P_i} for AS and P_i respectively, where $i = 1, 2, \cdots, n$.
- **Step 2. Training Parameter Initialization**
  AS first generates global model parameters W_g randomly. Then, AS selects the pruning number N_p and shuffling number N_s as the parameters of PTSP. Finally, AS sets the hyperparameters of model training, such as learning rate α, batch size B, and local training epoch E.

After the system initialization, TA transfers to offline and will not join the model training process. Then n participants and AS perform the following training process iteratively.

4.4.2. Model Perturbtion and Distribution

At each training round, AS first executes the above model perturbation function $\mathcal{P}(W_g, N_p, N_s)$ to obtain the perturbed global model parameters W_p. Then AS sends W_p to each $P_i \in \mathcal{A}_p$ and waits for their training completed.

4.4.3. Local Training and Encryption

In this phase, every P_i first performs local training. After that, AS sends the encryption parameters to participants who have completed training. Finally, every $P_i \in \mathcal{A}_f$ sends the encrypted local updates to AS.

- **Step 1. Local Data Training**
  After receiving the perturbed global model parameters W_p, participant $P_i \in \mathcal{A}_p$ uses its training data D_i to train W_p locally. Through E-epoch MB-SGD training, P_i obtains the local update ω_i.
- **Step 2. Encryption Parameter Distribution**
  When Step 1 is completed, P_i sends the training completion signal to AS. Once AS receives more than t signals from participants who have completed local data training, it adds them to a new list $\mathcal{A}_f$. Then, AS announces the total number of data $N_d = \sum_{P_i \in \mathcal{A}_f} N_i$ and sends the encryption parameter $\gamma_i = \prod_{P_j \in \mathcal{A}_f, j \neq i} \frac{j}{j-i}$ for each $P_i \in \mathcal{A}_f$,
  In particular, even if some participants fail to complete the local training or send a training completion signal due to equipment failure or connection interruption, as long as the number of remaining participants exceeds t, the rest of the training process can still proceed successfully.
- **Step 3. Local Update Encryption**
  After receiving γ_i, P_i performs $FTSA.Enc\ (PP, SK_{P_i}, \omega_i)$ function to calculate encrypted local update $[\![\omega_i]\!]$ and sends it to AS.

4.4.4. Secure Aggregation and Model Recovery

In this phase, AS aggregates the encrypted local updates of multiple participants as the aggregation update, which is further recovered to W_g for the next training round.

- **Step 1. Secure Aggregation**
  After receiving all encrypted local updates from $P_i \in \mathcal{A}_f$, AS executes function $FTSA.Agg(PP, [\![\omega_i]\!]_{P_i \in \mathcal{A}_f})$ to aggregate the encrypted local updates as $[\![\omega_g]\!]$, then computes $FTSA.Dec(PP, SK, [\![\omega_g]\!])$ to get aggregation update ω_g.
- **Step 2. Model Recovery**
  Finally, AS executes function $\mathcal{R}(\omega_g, R_s, N_p)$ to recover new global model parameters W_g, and updates the list of active participants $\mathcal{A}_p$ for the next training round.

Algorithm 3: Protocol Process of CORK

Input: local training data $\{D_i\}_{P_i \in \mathcal{A}_p}$.
Output: convergence model W_g.
TA:
1: Form $\mathcal{A}_p = \{P_1, P_2, \ldots, P_n\}$.
2: Select κ and $t (t < n)$.
3: $(PP, SK, \{SK_{P_i}\}_{i \in \mathcal{A}_p}) \leftarrow FTSA.Gen(\kappa, t, \mathcal{A}_p)$
4: Announce PP.
5: Send SK to AS.
6: Send SK_{P_i} to $P_i \in \mathcal{A}_p$.
AS:
7: Initialize W_g randomly.
8: Select N_s, N_p, α, B, and E.
9: **while** W_g doesn't converge **do**
10: $W_p, R_s \leftarrow \mathcal{P}(W_g, N_p, N_s)$.
11: Send W_p to $P_i \in \mathcal{A}_p$.
12: $\mathcal{A}_f = \{\}$.
$\boldsymbol{P_i} \in A_{\boldsymbol{p}}$:
13: Receive W_p from AS.
14: Train $\omega_i \leftarrow \frac{N_i}{N_d}$·MB-SGD$(E, \alpha, B, W_p, D_i)$.
15: Send training completion signal to AS.
AS:
16: Receive signal and add P_i to $\mathcal{A}_f$.
17: **Assert** $|\mathcal{A}_f| \geqslant t$.
18: **for** $P_i \in \mathcal{A}_f$ **do**
19: Calculate $\gamma_i = \prod_{P_j \in \mathcal{A}_f, j \neq i} \frac{j}{j-i}$.
20: Send γ_i to P_i.
$\boldsymbol{P_i} \in A_{\boldsymbol{f}}$:
21: Receive γ_i from AS.
22: Calculate $[\![\omega_i]\!] = FTSA.Enc(PP, SK_{P_i}, \omega_i)$.
23: Send $[\![\omega_i]\!]$ to AS.
AS:
24: Receive $[\![\omega_i]\!]$ from P_i.
25: **end for**
26: Calculate $[\![\omega_g]\!] = FTSA.Agg(PP, \{[\![\omega_i]\!]\}_{P_i \in \mathcal{A}_f})$.
27: Decrypt $\omega_g = FTSA.Dec(PP, SK, [\![\omega_g]\!])$.
28: $W_g \leftarrow \mathcal{R}(\omega_g, R_s, N_p)$.
29: $\mathcal{A}_p \leftarrow \mathcal{A}_f$.
30: **end while**

4.5. Correctness Analysis of CORK

In order to verify the correctness of CORK, we prove the following three theorems.

Theorem 1. At each training round, AS can obtain the correct aggregation update $\omega_g = \sum_{P_i \in \mathcal{A}_f} \omega_i$.

Proof. According to Eq. (2), local update ω_i is encrypted as

$$[\![\omega_i]\!] = g^{\omega_i} \cdot h^{r_i} \cdot SK_{P_i}^{(S-1)!\gamma_i} \bmod N^2.$$

Then, AS aggregates all encrypted local updates $\{[\![\omega_i]\!]\}_{P_i \in \mathcal{A}_f}$ as

$$\begin{aligned}[\![\omega_g]\!] &= \prod_{P_i \in \mathcal{A}_f} [\![\omega_i]\!] \bmod N^2 \\ &= g^{\sum_{P_i \in \mathcal{A}_f} \omega_i} \cdot h^r \cdot s^{q \sum_{P_i \in \mathcal{A}_f} (S-1)!f(i)\gamma_i} \bmod N^2,\end{aligned}$$

where $r = \sum_{P_i \in \mathcal{A}_f} r_i$, $\gamma_i = \prod_{P_j \in \mathcal{A}_f, j \neq i} \frac{j}{j-i}$, and $f(\cdot)$ is the polynomial generated by TA, as shown in Eq. (1).

Then, based on Shamir secret sharing and Lagrange interpolation formula, if the number of participants in $\mathcal{A}_f$ is more than threshold t,

$$\begin{aligned} [\![\omega_g]\!] &= g^{\sum_{P_i \in \mathcal{A}_f} \omega_i} \cdot h^r \cdot s^{q \cdot (S-1)! \cdot (0 \bmod p)} \\ &= g^{\sum_{P_i \in \mathcal{A}_f} \omega_i} \cdot h^r \cdot s^{kN}, \end{aligned} \tag{4}$$

where $k \in \mathbb{Z}_p$.

Finally, AS decrypts $[\![\omega_g]\!]$ as

$$\begin{aligned} \omega_g &= \frac{L([\![\omega_g]\!]^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \bmod N \bmod p' \\ &= \left(\sum_{P_i \in \mathcal{A}_f} \omega_i + rp'\right) \bmod p' \\ &= \sum_{P_i \in \mathcal{A}_f} \omega_i. \end{aligned} \tag{5}$$

Theorem 2. Even though some participants drop out during training, as long as the number of remaining participants is greater than t, the overall model training can proceed normally.

Proof. (Sketch) At the beginning of each training round, AS sends the global model parameters to all active participants. Once AS could receive more than t participants' local updates, according to Eq. (4) and Eq. (5), the aggregation update can be computed and decrypted correctly. And when entering the next training round, the remaining participants reorganize the list $\mathcal{A}_p$. Similarly, more than t local updates received can be aggregated correctly. And the rest may be deduced by analogy. Therefore, as long as the number of remaining participants is greater than t, the overall model training can proceed normally.

Theorem 3. The accuracy of the generated global model is not affected by the PTSP mechanism.

Proof. (Sketch) Function $\mathcal{P}(\cdot)$ in PTSP consists of two steps, neuron pruning, and neuron shuffling. At each training round, the neuron pruning operation removes and merges some neurons of the global model. Every pruning finds the two most similar neurons and merges them into one neuron. Since similar neurons are redundant and non-informative in the DNN model, neuron pruning doesn't affect the accuracy of the models. Besides, the neuron shuffling operation randomly shuffles the orders of neurons of the global model. According to the structure of the DNN model shown in Section 3.1, the training process is independent of the orders of neurons. So neuron shuffling doesn't affect the accuracy of the models either. In summary, function $\mathcal{P}(\cdot)$ in PTSP doesn't destroy the overall structure of the neural network and doesn't reduce the accuracy of the model at each training round.

5. Security Analysis

In this section, we analyze the security of CORK. Specifically, corresponding to the threat model and security requirements discussed in Section 2, we mainly focus on protecting the sensitive data in global model parameters and local updates from inference attacks.

5.1. Against the inference attack of AS

At each training round, AS can receive the local updates from participants and obtain the unperturbed global model parameters. On the one hand, from global model parameters, AS cannot analyze any sensitive information since AS doesn't have any auxiliary training data, which is proved in [10]. On the other hand, the local updates available to AS are encrypted in CORK. Therefore, in order to prove that CORK can resist the inference attack of AS, we just need to prove that AS cannot decrypt a single encrypted local update.

Theorem 4. During each training round, AS cannot decrypt $[\![\omega_i]\!]$ to obtain the local update ω_i.

Proof. (Sketch) In FTSA algorithm, we first split the public parameter N into multiple $f(i)$s for P_i in $\mathcal{A}_p$ by using the Shamir secret sharing. Then, the secret key of P_i is calculated as

$$SK_{P_i} = s^{q \cdot f(i)} \bmod N^2.$$

Finally, the local update of P_i is encrypted with SK_{P_i} as

$$[\![\omega_i]\!] = g^{\omega_i} \cdot h^{r_i} \cdot SK_{P_i}^{(S-1)!\gamma_i} \bmod N^2.$$

According to the Paillier cryptosystem, only when the third term in the above equation is recovered to the form like π^N, $[\![\omega_i]\!]$ can be decrypted with SK, where π is an arbitrary integer. As a result, the secret key SK of Paillier cryptosystem cannot decrypt ciphertexts encrypted with SK_{P_i}.

However, according to the Shamir secret sharing, when the local updates of more than $t-1$ participants are aggregated, the encrypted aggregation update is calculated as

$$[\![\omega_g]\!] = g^{\sum_{P_i \in \mathcal{A}_f} \omega_i} \cdot h^r \cdot s^{kN} \bmod N^2.$$

It can obviously be seen that the parameter N is recovered in the third term. According to Paillier cryptosystem, AS can decrypt it with SK to get the plaintext model parameters ω_g. Therefore, with our FTSA, SK can only be used to decrypt the aggregation update, where local updates of more than $t-1$ participants are aggregated, but not a single encrypted local update.

5.2. Against the inference attack of a participant

During each training round, if a participant could get the unperturbed global model parameters, it can get the aggregated local updates of others by calculating the difference between consecutive global model parameters, then it could infer other participants' sensitive information.

But in CORK, the global model parameters sent by AS are perturbed by our PTSP. As shown in Algorithm 2, model perturbation function $\mathcal{P}(\cdot)$ first prunes and merges some neurons of the global model, then shuffles the orders of neurons.

Detailedly, as shown in Fig. 5, the dotted line rectangles represent the removed weight parameters, which is impossible to be recovered after neuron pruning operation. And the red rectangles represent the merged weight parameters, whose values are the sum of two or more weights. As a result, it is also impossible to be recovered and it cannot be used to match the neuron positions of consecutive models. Besides, although there are very few remaining weights represented by the solid line rectangles, their values in consecutive models are also be changed by the training algorithm. Moreover, after the neuron shuffling operation, the positions of neurons are exchanged randomly. In short, the orders and values of global model parameters are changed significantly, which cannot be recovered by the participants.

Therefore, an honest-but-curious participant cannot match the positions of the same neurons in consecutive global models. As a result, it cannot get the aggregation of others' local updates, and our scheme can protect the sensitive data against the inference attack of an honest-but-curious participant mentioned in Section 3.3.

5.3. Against the inference attack of collusive AS and participants

If AS and some honest-but-curious participants collude to infer other honest participants' sensitive data, they can control AS's secret key SK and some honest-but-curious participants' secret key $\{SK_{P_i}\}_{P_i \in \mathcal{A}_c}$, where $\mathcal{A}_c$ is the list of collusive participants. To ensure that CORK can resist the collusive inference attack, we need to prove that they cannot decrypt a single local update of any participant as long as $|\mathcal{A}_c| < t$.

Theorem 5. The encrypted local update $[\![\omega_i]\!]$ cannot be decrypted with SK and $\{SK_{P_i}\}_{P_i \in \mathcal{A}_c}$ when $|\mathcal{A}_c| < t$.

Proof. (Sketch) The local update of a participant is encrypted as

$$[\![\omega_i]\!] = g^{\omega_i} \cdot h^{r_i} \cdot SK_{P_i}^{(S-1)!\gamma_i} \bmod N^2.$$

Based on Paillier cryptosystem, only when the exponent of the third term is recovered to public parameter N, $[\![\omega_i]\!]$ can be decrypted with SK.

However, according to the Shamir secret sharing, only when the number of SK_{P_i} is equal to or more than t, we can recover the exponent of the third term to N and decrypt $[\![\omega_i]\!]$.

Therefore, if $|\mathcal{A}_c| < t$, the number of SK_{P_i} is less than t, and the encrypted local update $[\![\omega_i]\!]$ cannot be decrypted.

6. Performance Evaluation

In this section, we evaluate and analyze the performance of our CORK in terms of accuracy, computation cost, and communication overhead, and make a comparison with a multiparty deep learning scheme called $\propto$MDL [12]. Based on asynchronous optimization, homomorphic encryption, and threshold secret sharing, $\propto$MDL can ensure that participants learn the global model only if a sufficient number of local updates are aggregated. Specifically, the ElGamal encryption is used in $\propto$MDL to achieve the aggregation of encrypted local updates. Moreover, based on the Shamir secret sharing, the aggregation process can also tolerate participant dropping out, but the aggregation server can obtain the global model parameters only when it has been decrypted by all active participants.

6.1. Evaluation Environment

In order to measure the performance of CORK, we perform our experiments in Python 3.7 on a workstation running Windows 10 system with Intel Core i5-9400H 2.50 GHz CPU and 24.0 GB RAM, and we use multiple processes to simulate different parts in CORK. Besides, to make the comparison between our CORK and $\propto$MDL more intuitive and convincing, we also implement $\propto$MDL in the same experimental environment. Meanwhile, the handwritten digits recognition (MNIST) [27] and image recognition (CIFAR-10) [28] datasets are used as the benchmark datasets to evaluate the model accuracy, and the synthetic datasets with different sizes are generated to test the factors affecting the computation cost and communication overhead. The details of the two real datasets are described as follows.

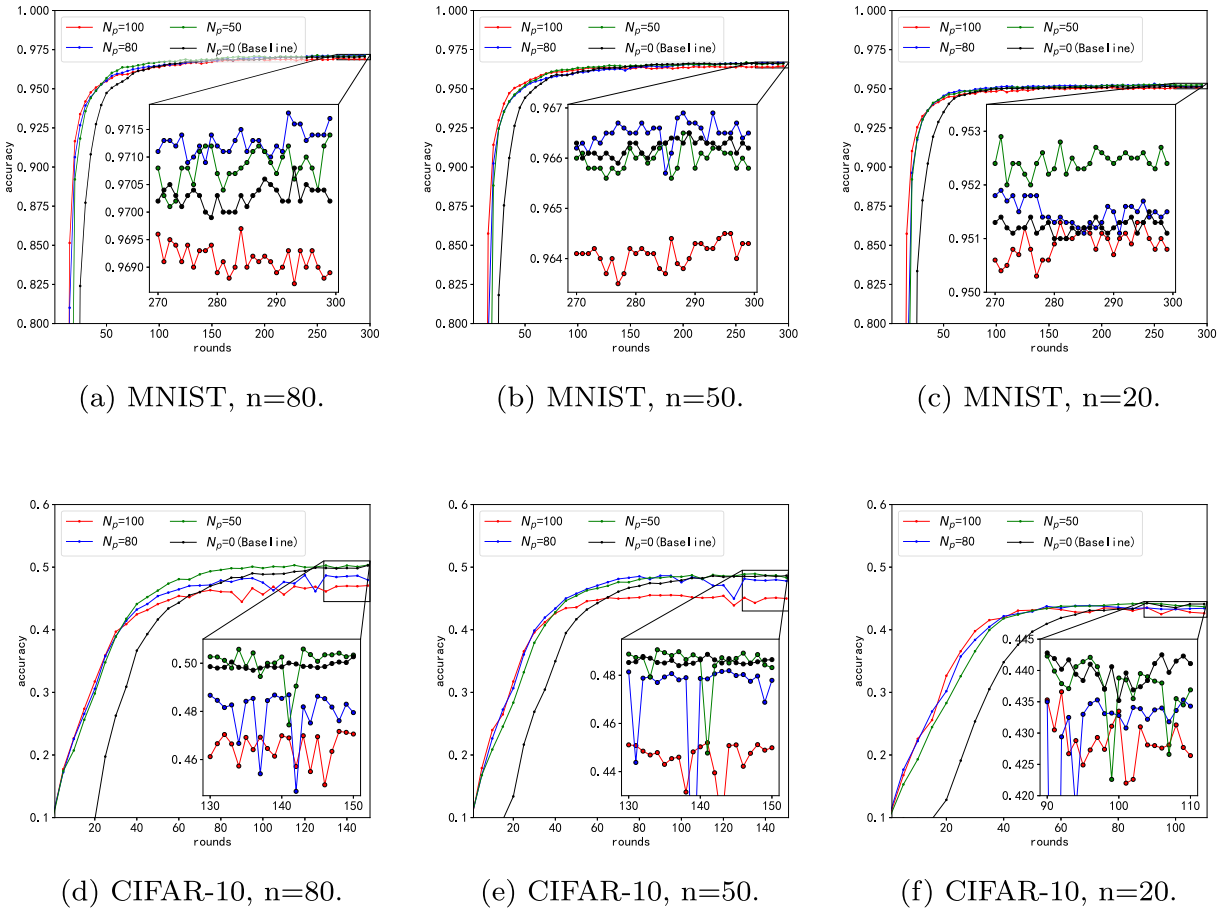
- The MNIST dataset consists of 60000 training and 10000 testing data of the handwritten digit images from 0 to 9. The digits are size-normalized and centered in a fixed-size 28×28 image, and the output size is 1×10. We divide 60000 training data evenly to 100 participants, and each participant obtains 600 training data. Each participant performs 5-epoch MB-SGD at each training round, and the learning rate and batch size are set to 0.01 and 32 respectively.
- The CIFAR-10 dataset contains 60000 color images in 10 classes (e.g., bird, frog, ship, etc.), with 6000 images per class. The dataset is divided into five training batches and one test batch, each with 10000 data. Moreover, the input and output size of each data is $32 \times 32 \times 3$ and 1×10 respectively. We divide 50000 training data evenly into 100 participants, and each participant obtains 500 training data. Each participant performs 3-epoch MB-SGD at each training round, and the learning rate and batch size are set to 0.1 and 32 respectively.

The goal of our task with MNIST and CIFAR-10 datasets is to train DNN models that can recognize 10 different handwritten digits and image classifications. The model architecture in both MNIST and CIFAR-10 tasks is a fully-connected DNN with two hidden layers containing 256 neurons, and the rectified linear unit (ReLU) is used as the activation function.

6.2. Accuracy

To evaluate the model accuracy of CORK, we perform model training when the number of participants n is $20, 50$, and 80. And for the training of a fixed participant number, we compare the convergence curves when the pruning number N_p is set to $0, 50, 80$, and 100 (for simplicity, we prune the same number of neurons in every layer, and a pruning number of 0 means that the model is not be pruned, which is equivalent to the model training with federated averaging). Since the shuffling number N_s will not have any impact on model training, we set it to a fixed value of 1000. Besides, the model accuracy is measured by the probability of correct prediction on the testing dataset.

The experiment results are shown in Fig. 6. The figures (a) to (c) are the model convergence curves for different numbers of participants with the MNIST dataset, and the figures (d) to (f) are the corresponding curves with the CIFAR-10 dataset. By comparing the convergence of the models with the same datasets, we find that the accuracy of the convergence models will raise with the increase of participant numbers. And the model convergence curves with different pruning numbers are shown in a single subfigure. For example, in Fig. 6a, we observe that the model accuracy of $N_p = 50, 80$ is higher than the model accuracy without pruning, and it is slightly lower than the model accuracy without pruning when $N_p = 100$, which shows our CORK will hardly affect the model accuracy when compared with the federated averaging. Furthermore, we find that the model converges faster when the pruning operation is added to the model training. In other words, the model in



(a) MNIST, n=80. (b) MNIST, n=50. (c) MNIST, n=20.

(d) CIFAR-10, n=80. (e) CIFAR-10, n=50. (f) CIFAR-10, n=20.

Fig. 6. Model accuracy of CORK.

Table 2
Accuracy of Convergence Models for Different Datasets, Participant Numbers, and Pruning Numbers

dataset	participant number	pruning number			
		0	50	80	100
MNIST	20	95.16%	95.31%	**95.32%**	95.13%
	50	96.65%	96.69%	**96.78%**	96.46%
	80	97.08%	97.25%	**97.30%**	96.97%
CIFAR-10	20	44.28%	**44.42%**	44.15%	43.78%
	50	48.95%	**49.06%**	48.97%	45.75%
	80	50.05%	**50.60%**	48.91%	47.36%

CORK converges faster than in federated averaging. In order to show the experimental results more clearly, the accuracy of convergence models can be seen in Table 2.

6.3. Computation Cost

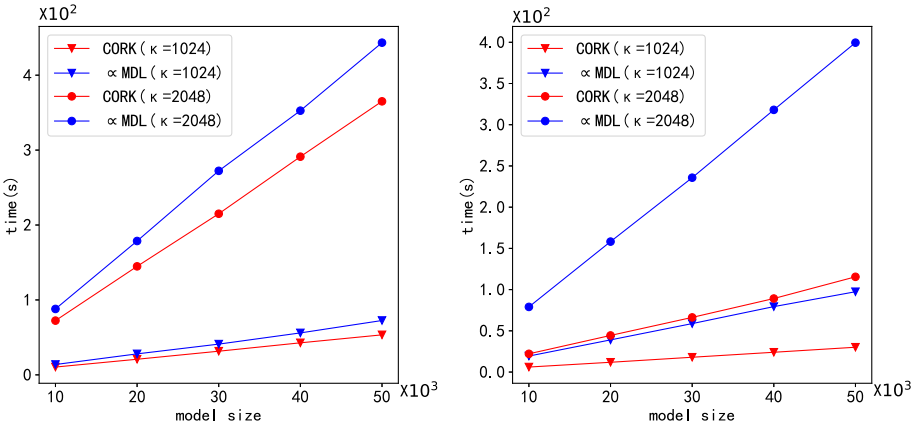
Next, we analyze and evaluate the computation cost of CORK for both AS and participants, and compare the evaluation result to $\propto$MDL.

6.3.1. Theoretical Analysis

In our analysis, we ignore some simple computations, because they have little effect on the computation cost. Besides, since the MB-SGD algorithm is executed over plaintext, the computation cost of it is not considered. As a result, we only con-

Table 3
Computation Cost Comparison in Theoretical Analysis.

	CORK	$\propto$MDL
AS	$P_c \cdot S_m((n-1)\tau_{m2} + \tau_{m1} + \tau_{e2})$	$S_m(3(n-1)\tau_{m1} + \tau_{e1})$
P_i	$P_c \cdot S_m(3\tau_{e2} + 2\tau_{m2})$	$S_m(\tau_{m1} + 4\tau_{e1})$



(a) Computaion cost of P_i. (b) Computaion cost of AS.

Fig. 7. Computation cost comparison in experimental evaluation.

sider the time spent on two kinds of time-consuming computations: modular multiplication and modular exponentiation. For simplicity, we set that modular multiplications in $\mathbb{Z}_N$ and $\mathbb{Z}_{N^2}$ cost τ_{m1} and τ_{m2} respectively, and modular exponentiations in $\mathbb{Z}_N$ and $\mathbb{Z}_{N^2}$ cost τ_{e1} and τ_{e2} respectively. In addition, consistent with the above, we use n to represent the number of training participants. And S_m represents the size of model parameters, P_c means model compression rate caused by our PTSP.

At each training round of CORK, participants encrypt their local updates through FTSA algorithm. And for each participant, the time cost is $P_c \cdot S_m \cdot (3\tau_{e2} + 2\tau_{m2})$. Then, AS aggregates these encrypted local updates, which costs $P_c \cdot S_m \cdot (n-1) \cdot \tau_{m2}$. Besides, the decryption operation costs $P_c \cdot S_m \cdot (\tau_{m1} + \tau_{e2})$. So the total cost of AS is $P_c \cdot S_m \cdot ((n-1) \cdot \tau_{m2} + \tau_{m1} + \tau_{e2})$.

For $\propto$MDL, every participant encrypts its local update through a two-step calculation at each training round. In the first step, the time cost is $S_m \cdot (\tau_{m1} + 3\tau_{e1})$. And in the second step, $S_m \cdot \tau_{e1}$ is cost. So the total cost of each participant is $S_m \cdot (\tau_{m1} + 4\tau_{e1})$. And for AS, it also takes two steps to aggregate the local updates. In the first step, the time cost is $S_m \cdot ((n-1) \cdot \tau_{m1} + \tau_{e1})$. And in the second step, $S_m \cdot 2(n-1) \cdot \tau_{m1}$ is cost. So AS's total time cost is $S_m \cdot (3 \cdot (n-1) \cdot \tau_{m1} + \tau_{e1})$.

We compare the theoretical computation cost between our CORK and $\propto$MDL in Table 3. From the table, we can see that for both AS and P_i, the computation cost of CORK for a single parameter is lower than $\propto$MDL. Besides, since CORK prunes the global model by neuron pruning operation at each training round, the size of the model parameters is smaller than $\propto$MDL. In CORK, the pruning S_m rate is usually lower than 70%, so the computation cost is much less than $\propto$MDL. In general, CORK is more efficient than $\propto$MDL in computation cost.

6.3.2. Experimental Evaluation

We also evaluate the actual computation cost of CORK and $\propto$MDL, and plot the experimental results in Fig. 7. Specifically, we evaluate the conditions when security parameter $\kappa = 1024$ and 2048 respectively, the number of participants $n = 80$, and the size of global model parameters S_m varying from 10000 to 50000. Besides, in CORK, we set a conservative compression rate $P_c = 0.7$.

By observing the experimental results, we find that the computation cost of both CORK and $\propto$MDL linearly increases with the model size. But CORK is more efficient than $\propto$MDL, and it means that our scheme is more practical in application.

6.4. Communication Overhead

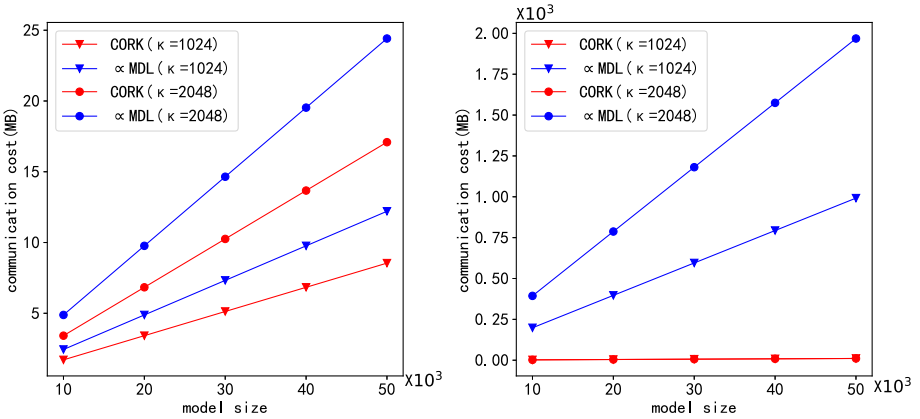
We analyze and evaluate the communication overhead of CORK in this section, and make a comparison with $\propto$MDL.

6.4.1. Theoretical Analysis

In the analysis of communication overhead, same as above, S_m represents the size of model parameters, P_c means model compression rate caused by pruning operation in PTSP, κ is the security parameter, and n is the number of participants. Besides, we set that a single plaintext model parameter costs ζ bits.

Table 4
Communication Overhead Comparison in Theoretical Analysis.

	CORK	∝MDL
AS	$n \cdot \zeta(S_m \cdot P_c + 1)$	$S_m \cdot n(\zeta + 2\kappa)$
P_i	$S_m \cdot P_c \cdot 2\kappa$	$S_m \cdot 2\kappa$



(a) Communication overhead of P_i. (b) Communication overhead of SP.

Fig. 8. Communication overhead comparison in experimental evaluation.

At each training round of CORK, AS sends the perturbed global model parameters to each participant, which needs $S_m \cdot P_c \cdot n \cdot \zeta$ bits. And each participant returns its encrypted local update, which costs $S_m \cdot P_c \cdot 2\kappa$ bits. Besides, the encryption parameters sent by AS costs additional $n \cdot \zeta$ bits. Therefore, the communication overhead of AS and a single participant is $n \cdot \zeta(S_m \cdot P_c + 1)$ bits and $S_m \cdot P_c \cdot 2\kappa$ bits respectively.

At each training round of ∝MDL, AS firstly sends the global model parameters to each participant, which costs $S_m \cdot n \cdot \zeta$ bits. Then the return encrypted local update from each participant costs $S_m \cdot 2\kappa$ bits. Besides, to decrypt the aggregation update, ∝MDL needs to communicate additional $S_m \cdot n \cdot 2\kappa$ bits. So the communication of AS and a single participant costs $S_m \cdot n(\zeta + 2\kappa)$ bits and $S_m \cdot 2\kappa$ bits respectively.

We compare the theoretical communication overhead between our CORK and ∝MDL in Table 4. From the table, for both AS and P_i, we see that our communication overhead is less than ∝MDL even without the neuron pruning operation. And after pruning, our communication overhead is reduced by at least 30% on the original overhead. Therefore, CORK is more efficient than ∝MDL in communication overhead.

6.4.2. Experimental Evaluation

We also estimate the actual communication overhead of CORK and ∝MDL, and plot the experimental results in Fig. 8. Specifically, we set security parameters $\kappa = 1024$ and 2048 respectively, the number of participants $n = 80$, the length of a single plaintext model parameter $\zeta = 32$ bits, and the size of global model parameters S_m varying from 10000 to 50000. Besides, in CORK, we set a conservative compression rate $P_c = 0.7$.

From the experimental result, we can find that the communication overhead linearly raises with the increase of the model size in both our CORK and ∝MDL. But the slope of CORK is much smaller than that of ∝MDL, which means that CORK is more suitable for the training of large models.

7. Related Work

In this section, we introduce some related works about privacy-preserving federated learning for DNN. And we classify the existing schemes into two categories: based on SA [29,12,13,30–33,16] or DP [34,18,35,36,19,20].

7.1. Privacy-preserving Federated Learning via SA

The SA algorithm can achieve that the aggregation server obtains the aggregation of participants' data without knowing their individual data.

Table 5
Function Comparison between CORK and Other Schemes.

	[12]	[13]	[7]	[32]	[18]	CORK
Protected global model	×	×	✓	×	✓	✓
Protected local update	✓	✓	✓	✓	✓	✓
Drop-tolerant	✓	✓	×	✓	×	✓
Lossless	✓	✓	×	✓	×	✓
High efficiency	×	✓	✓	✓	✓	✓

Chan et al. [29] proposed an algorithm to achieve privacy-preserving data aggregation with an untrusted aggregator, which can efficiently resist user failure and support dynamic joins and leaves. Zhang et al. [12] proposed an SA algorithm for model parameters based on additive homomorphic cryptosystem and threshold secret sharing technology, which is still available when a few participants drop out. Corrigan et al. [31] presented an efficient SA algorithm based on the secret-shared non-interactive proofs, which can achieve a least-squares regression on high-dimensional data. Xu et al. [32] designed a secure and verifiable aggregation algorithm based on double-masking protocol and proof of integrity. It can not only achieve SA but also allow participants to verify the integrity of the aggregation result. By utilizing masking encryption, Zhang et al. [16] designed a SA protocol to compute the product of n numbers privately and efficiently. Moreover, SA can also be achieved by using trusted execution environment (TEE), which is designed by Lie et al. [30].

SA-based schemes don't affect the accuracy of models and can provide a high privacy guarantee; however, in these schemes, the local training of participants still proceed on the plaintext global model parameters, and they cannot prevent the inference attack of an honest-but-curious participant.

7.2. Privacy-preserving Federated Learning via DP

The DP mechanism can prevent any participant from trying to infer the sensitive data of others by exploiting the global model parameters, which is achieved by adding random noises on local updates to hide any single participant's characteristics.

Abadi et al. [18] proposed a differential private SGD algorithm by adding additive noises to the local updates and develop an advanced analysis method of privacy budget. Mohassel et al. [35] proposed a distributed privacy-preserving machine learning framework based on the local DP mechanism. In the client–server asynchronous communication mode, high-quality machine learning models can be trained while satisfying DP. Pihur et al. [36] proposed a privacy-preserving time series data aggregation scheme according to distributed DP mechanism. The scheme realizes the secure aggregation of time series data by disturbing the data of multiple participants in different time series. Bhowmick et al. [19] designed an optimal local differential private mechanism for large-scale statistical learning by limiting the ability of adversaries. Li et al. [20] proposed a personalized local differentially private mechanism in meta-learning, which can provide provable learning guarantees in convex settings.

DP-based schemes can afford strong privacy guarantees, but it inevitably reduces the accuracy of the trained models due to the added noises.

Different from the above schemes, our proposed CORK achieves privacy-preserving and lossless federated learning for DNN. On the one hand, different from secure aggregation schemes, CORK can protect sensitive data well in both local updates and global model parameters from inference attacks. On the other hand, compared with the DP-based schemes, CORK doesn't cause a loss of model accuracy. Detailedly, we compare the functions between our CORK and other existing schemes in Table 5.

8. Conclusion

In this paper, we proposed CORK, a privacy-preserving and lossless federated learning scheme for DNN. Based on our proposed drop-tolerant secure aggregation algorithm FTSA, on the one hand, CORK can ensure that local updates of participants are confidential to AS, on the other hand, could keep the normal training process even though some participants drop out. Besides, by applying our lossless model perturbation mechanism PTSP, under the premise of lossless model accuracy, a participant cannot obtain the aggregation update of others from the perturbed global model parameters. Finally, detailed security analysis shows that CORK can protect sensitive data well against the inference attacks of AS, participants, and their collusion, and extensive experiments are performed to show the efficiency of CORK in both computation and communication overhead.

CRediT authorship contribution statement

Jiaqi Zhao: Conceptualization, Writing - original draft, Software. **Hui Zhu:** Project administration, Funding acquisition. **Fengwei Wang:** Writing - review & editing. **Rongxing Lu:** Methodology. **Hui Li:** Supervision. **Jingwei Tu:** Funding acquisition. **Jie Shen:** Funding acquisition.

Declaration of Competing Interest

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.

Acknowledgment

This work was supported by National Natural Science Foundation of China (61972304, 61932015), Science Foundation of the Ministry of Education (MCM20200101), and Shaanxi Provincial Key Research and Development Program (2020ZDLGY08-04).

References

[1] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, IEEE Comput. Intell. Mag. 13 (3) (2018) 55–75, https://doi.org/10.1109/MCI.2018.2840738.
[2] W. Zong, G. Huang, Face recognition based on extreme learning machine, Neurocomputing 74 (16) (2011) 2541–2551, https://doi.org/10.1016/j.neucom.2010.12.041.
[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T.P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, Nat. 550 (7676) (2017) 354–359, https://doi.org/10.1038/nature24270.
[4] J. Konecný, H.B. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, CoRR abs/1610.05492. arXiv:1610.05492..
[5] N. Papernot, P.D. McDaniel, A. Sinha, M.P. Wellman, Sok: Security and privacy in machine learning, EuroS&P, IEEE (2018) 399–414, https://doi.org/10.1109/EuroSP.2018.00035.
[6] L. Lyu, H. Yu, Q. Yang, Threats to federated learning: A survey, CoRR abs/2003.02133. arXiv:2003.02133..
[7] 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, https://doi.org/10.1109/TIFS.2017.2787987.
[8] L. Zhu, S. Han, Deep leakage from gradients, in: NeurIPS, Morgan Kaufmann, 2019, pp. 14747–14756. doi:10.1007/978-3-030-63076-8_2..
[9] B. Zhao, K.R. Mopuri, H. Bilen, idlg: Improved deep leakage from gradients, CoRR abs/2001.02610. arXiv:2001.02610..
[10] L. Melis, C. Song, E.D. Cristofaro, V. Shmatikov, Exploiting unintended feature leakage in collaborative learning, in: IEEE Symposium on Security and Privacy, IEEE, 2019, pp. 691–706, https://doi.org/10.1109/SP.2019.00029.
[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for federated learning on user-held data, CoRR abs/1611.04482. arXiv:1611.04482..
[12] X. Zhang, S. Ji, H. Wang, T. Wang, Private, yet practical, multiparty deep learning, in: ICDCS, IEEE Computer Society, 2017, pp. 1442–1452. doi:10.1109/ICDCS.2017.215..
[13] 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: CCS, ACM, 2017, pp. 1175–1191. doi:10.1145/3133956.3133982..
[14] F. Wang, H. Zhu, R. Lu, Y. Zheng, H. Li, Achieve efficient and privacy-preserving disease risk assessment over multi-outsourced vertical datasets, IEEE Trans. Dependable Secur. Comput. doi:10.1109/TDSC.2020.3026631..
[15] F. Wang, H. Zhu, R. Lu, Y. Zheng, H. Li, A privacy-preserving and non-interactive federated learning scheme for regression training with gradient descent, Inf. Sci. 552 (2021) 183–200, https://doi.org/10.1016/j.ins.2020.12.007.
[16] X. Zhang, X. Chen, H. Yan, Y. Xiang, Privacy-preserving and verifiable online crowdsourcing with worker updates, Inf. Sci. 548 (2021) 212–232, https://doi.org/10.1016/j.ins.2020.10.010.
[17] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: CCS, ACM, 2015, pp. 1310–1321. doi:10.1145/2810103.2813687..
[18] M. Abadi, A. Chu, I.J. Goodfellow, H.B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: CCS, ACM, 2016, pp. 308–318.
[19] A. Bhowmick, J.C. Duchi, J. Freudiger, G. Kapoor, R. Rogers, Protection against reconstruction and its applications in private federated learning, CoRR abs/1812.00984. arXiv:1812.00984..
[20] J. Li, M. Khodak, S. Caldas, A. Talwalkar, Differentially private meta-learning, CoRR abs/1909.05830. arXiv:1909.05830..
[21] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117, https://doi.org/10.1016/j.neunet.2014.09.003.
[22] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, E. Prouff, Privacy-preserving classification on deep neural network, IACR Cryptol. ePrint Arch. (2017) 1–18, http://eprint.iacr.org/2017/035.
[23] S.J. Hanson, L.Y. Pratt, Comparing biases for minimal network construction with back-propagation, in: D.S. Touretzky (Ed.), NIPS, Morgan Kaufmann, 1988, pp. 177–185. doi:10.5555/2969735.2969756..
[24] S. Srinivas, R.V. Babu, Data-free parameter pruning for deep neural networks, in: BMVC, BMVA Press, 2015, pp. 31.1–31.12. doi:10.5244/C.29.31..
[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, CoRR abs/1602.05629. arXiv:1602.05629..
[26] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: EUROCRYPT, Vol. 1592 of Lecture Notes in Computer Science, Springer, 1999, pp. 223–238. doi:10.1007/3-540-48910-X_16..
[27] Y. LeCun, The mnist database of handwritten digits., http://yann.lecun.com/exdb/mnist/ (1998)..
[28] A. Krizhevsky, V. Nair, G. Hinton, The cifar-10 dataset., http://www.cs.toronto.edu/kriz/cifar.html (2014)..
[29] T.H. Chan, E. Shi, D. Song, Privacy-preserving stream aggregation with fault tolerance 7397 (2012) 200–214. doi:10.1007/978-3-642-32946-3_15..
[30] D. Lie, P. Maniatis, Glimmers: Resolving the privacy/trust quagmire, in: HotOS, ACM, 2017, pp. 94–99. doi:10.1145/3102980.3102996..
[31] H. Corrigan-Gibbs, D. Boneh, Prio: Private, robust, and scalable computation of aggregate statistics, in: NSDI, USENIX Association, 2017, pp. 259–282. doi:10.5555/3154630.3154652..
[32] G. Xu, H. Li, S. Liu, K. Yang, X. Lin, Verifynet: Secure and verifiable federated learning, IEEE Trans. Inf. Forensics Secur. 15 (2020) 911–926, https://doi.org/10.1109/TIFS.2019.2929409.

[33] X. Zhang, X. Chen, J.K. Liu, Y. Xiang, Deeppar and deepdpa: Privacy preserving and asynchronous deep learning for industrial iot, IEEE Trans. Ind. Informatics 16 (3) (2020) 2081–2090, https://doi.org/10.1109/TII.2019.2941244.
[34] C. Dwork, Differential privacy: A survey of results, in: TAMC, Vol. 4978, Springer, 2008, pp. 1–19. doi:10.1007/978-3-540-79228-4_1..
[35] P. Mohassel, Y. Zhang, Secureml: A system for scalable privacy-preserving machine learning, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, 2017, pp. 19–38. doi:10.1109/SP.2017.12..
[36] V. Pihur, A. Korolova, F. Liu, S. Sankuratripati, M. Yung, D. Huang, R. Zeng, Differentially-private draw and discard machine learning, CoRR abs/1807.04369. arXiv:1807.04369..