Received: 12 May 2022 | Accepted: 1 August 2022

DOI: 10.1002/int.22993

RESEARCH ARTICLE

WILEY

Robust privacy-preserving federated learning framework for IoT devices

Zhaoyang Han | **Lu Zhou** | **Chunpeng Ge** | **Juan Li** | **Zhe Liu**

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

Correspondence
Lu Zhou, College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Jiangjun St No. 29, 210016 Nanjing, China.
Email: lu.zhou@nuaa.edu.cn

Funding information
National Key R&D Program of China, Grant/Award Number: 2020AAA010770; National Natural Science Foundation of China, Grant/Award Numbers: 62076125, U20B2049, U20B2050

Abstract

Federated Learning (FL) is a framework where multiple parties can train a model jointly without sharing private data. Private information protection is a critical problem in FL. However, the communication overheads of existing solutions are too heavy for IoT devices in resource-constrained environments. Additionally, they cannot ensure robustness when IoT devices become offline. In this paper, Democratic Federated Learning (DemoFL) is proposed, which is a privacy-preserving FL framework that has sufficiently low communication overheads. DemoFL involves a consensus module to ensure the system is robust. It also utilizes a tree structure to reduce the time communication overheads and realizes high robustness without reducing accuracy. The proposed algorithm reduces the communication complexity of aggregation at training by M times, M being a controllable parameter. Sufficient experiments have been conducted to evaluate the efficiency of the proposed method. The experimental results also demonstrate the practicality of the proposed framework for IoT devices in unstable environments.

KEYWORDS
federated learning, machine learning, multi-party computation, resource-constrained devices, secure aggregation

1 | INTRODUCTION

Federated Learning (FL)[1] has gained a lot of attention as a way to jointly train machine learning models without revealing specific data about the participants. FL develops rapidly with an increasing number of applications. For example, Google's keyboard query suggestions[2] project is an effective application of FL, which trains a model that can predict the input of users precisely without uploading their private data. It is also widely applied for Internet of Things (IoT) devices now.[3] Figure 1 illustrates a typical structure of FL. As depicted in the figure, the participants (usually IoT devices) receive a global model from the aggregation center and train their models locally based on their data. When the training process is finished, each participant sends the parameters (or gradients) of their models to the center while the center runs a designed aggregation algorithm to compute the global model based on the received parameters. In such frameworks, participants are not required to share their private data directly, and thus the privacy of participants is to some extent protected.

However, there are some issues in the present FL frameworks. The first issue is privacy. Recent work points out that attackers can establish inference attacks[4–7] to reveal the private training data if the parameters/gradients in the training process are leaked. For example, the attackers may train shadow models on the predictive results of the targeted model to gain knowledge about data sources.[4] The second issue is efficiency. To solve the privacy problem, many secure aggregation protocols[8–11] based on cryptographic techniques have been proposed. These protocols are all based on secure multi-party computation (MPC), Homomorphic Encryption (HE), or Differential Privacy (DP). Since HE-based methods suffer from low efficiency which is hardly acceptable in FL,[12] and DP-based methods are barely satisfying,[13] MPC-based methods are the most widely adopted. However, applying MPC directly in secure aggregation protocols will cause high communication overheads.

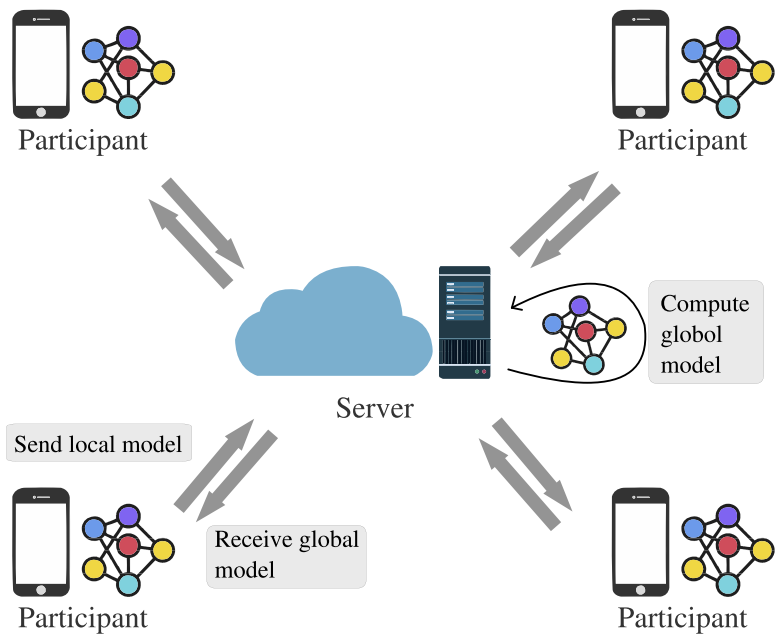

FIGURE 1 The structure of FedAvg[1] algorithm [Color figure can be viewed at wileyonlinelibrary.com]

For instance, the first secure aggregation algorithm SecAgg[14] requires secret key negotiation between all participants, and its handling of dropouts requires a lot of additional data exchange. This leads to the third issue of practicality and robustness. To improve communication efficiency, some work stratifies participants to reduce the number of communications by joining sub-learning groups.[15,16] However, the previous methods are not maintained for topology changes of the participants. For example, if the physical location layout of the IoT device changes significantly, the learning committee in the two-phase method[15] needs to be reorganized, resulting in additional overhead. In IoT environments (such as Low-Power Wide-Area Network (LPWAN)[17] environments), there is a need to establish FL algorithms among resource-constrained devices in addition to devices such as personal computers and high-performance smartphones. These devices may be deployed over wide areas (e.g., sensors in mountains or railroads) and have a low energy source (e.g., batteries). In many scenarios, the end devices (such as drones, smart vehicles, etc.) are required to be able to move and change their locations frequently. And these devices may have different work schedules. These facts indicate that the devices cannot establish a long-term FL consistently. Previous works do not consider the situations where the topology of participants may change. Employing traditional MPC-based methods directly in an FL system deployed in such environments faces the problem of low efficiency and instability.

In conclusion, previous work does not fully consider IoT devices in resource-constrained environments. Applying prior FL frameworks in resource-constrained environments may suffer from low efficiency, reduced accuracy, or complicated structures. These issues motivate us to design a privacy-preserving, communication-efficient, and robust FL framework. More specifically, we aim to design an FL framework that:

- protects users' privacy against semi-honest adversaries by employing cryptography techniques with a low communication overhead,
- is robust in resource-constrained environments where participants have limited ability and may change their positions and be in dormant status sometimes, and
- has little influence on the accuracy of the trained model.

Realizing such an FL framework is nontrivial. First, the number of training participants may be large while these participants have quite poor network conditions. As mentioned before, some participants need to communicate with all others several times in each round. The large communication volume and the limited resources cause a severe problem. Second, it is common for edge devices not to have a central server between them, while at the same time these devices may suddenly fall into a dormant state, which means that the topology of FL participants may change frequently. It is difficult for the architecture-less traditional FL frameworks to perceive and handle such member changes.

To tackle these problems, we propose Democratic Federated Learning (DemoFL), which is a privacy-preserving and communication-efficient FL framework. DemoFL uses a tree-structured organization of participants to greatly reduce the communication overheads for each FL participant. The communication cost can be reduced by $\frac{n}{m}$ times, where n is the number of participants and m is the maximum number of children of a node in the tree. DemoFL also has a consensus system as a submodule to maintain the tree structure, which is modified from the famous consensus algorithm Raft.[18]

1.1 | Our contribution

- We propose DemoFL, a novel FL framework that protects users' privacy. DemoFL adapts a smart additive secret sharing protocol to realize MPC and reduce the communication overheads from $O(N)$ to $O\left(\frac{N}{M}\right)$ by utilizing the proposed tree structure design.
- We design a consensus scheme to achieve high robustness in DemoFL. The consensus scheme helps to handle unexpected situations where a participant node is offline or the formation of an FL process changes while causing negligible overheads and provides high robustness for the FL system.
- We conduct sufficient experiments and verify that DemoFL has a satisfactory efficiency and robustness without reducing the accuracy of the trained model.

1.2 | Road map

In Section 2 we introduce the background and give definitions of related techniques. Section 3 illustrates our proposed framework including the attack model. Evaluations for efficiency are stated in Section 4, followed with experimental results in Section 5. Next, we introduce related work in Section 6. Finally, we give the conclusion and future expectations in Section 7.

2 | BACKGROUND

2.1 | Secure aggregation in FL

FL enables a number of participants to jointly train a machine learning model. It can be categorized into horizontal, vertical, and other types.[19] In this paper, we focus on the horizontal FL, where the participants have the same features on data sets. We take FedAvg[1] as an example and briefly introduce the working paradigm of FL: Denote the set of participants by P, the private data set possessed by the ith participant P_i by D_i. The purpose of the participants is to train a model M based on the private datasets. The training process usually consists of many rounds. In each round, each user will train the model based on some batches of its private data set and record the parameters from stochastic gradient descent (SGD) algorithm, which is used to update models. Denote the ith participant's local parameter in the t-th round by W_i^t. When all participants have trained their model in the t-th round, all the recorded parameters will be sent to a center S for aggregation. S executes an aggregation algorithm Agg to compute the global model's parameter $Agg(W_1^t, W_2^t, ..., W_n^t)$, where n is the total number of participants. FedAvg is still the most popular aggregation method.

Even in FL, there are still privacy issues. Some works point out that leaked parameters of FL models may be exploited by attackers to reveal the original data,[4–7] and the intermediate parameters in FL frameworks should also be protected. Due to the high computational complexity of HE and the unsatisfying privacy of DP, Secure MPC is more popular to protect FL parameters. MPC is a branch of cryptography that enables several parties to compute a particular function without leaking their own data (inputs). Suppose there are n participants who want to compute a function f with their private data, and the i-th participant has its private data a_i. Their goal is to compute $f(a_1, a_2, ..., a_n)$ without sharing the private data to others, i.e., a party P_i will have no idea about parameter $a_j (j \neq i)$ after MPC process. The form of MPC fits the aggregation algorithm in FL

appropriately. There are many ways to implement MPC, such as garbled circuits,[20] secret sharing,[21] and oblivious transfer.[22] Considering the scalability and computation complexity of these methods, secret sharing can be the most applicable approach to protect privacy in resource-constrained.[23] A secret sharing scheme involves a secret s, a set of n parties, and a collection A of subsets of parties. s will be split into pieces and each party has one share of it. The secret sharing scheme ensures any subset in A can reconstruct s.

In the FL framework, secure addition is the most important arithmetic, because all the aggregation algorithm needs to compute is the weighted average of the local parameters, that is, only the sum of all parameters and the sum of the number of training data. In Google's SecAgg[14] algorithm, additive secret sharing is implemented by generating random masks. Its overview framework is shown in Figure 2. However, SecAgg has unsatisfactory communication overheads because it requires a participant to send secret pieces to all other participants in each round. SecAgg works in most industrial scenarios, but in resource-constrained scenarios, its unrefined and optimized communication model is insufficient.

2.2 | Consensus algorithms

In a distributed or multiagent system, there is always a problem with consensus, that is, the parties of such systems always need to achieve agreement on a certain value. Consensus algorithms are adopted to address this problem. It is widely used in various areas such as blockchain[24] and distributed storage systems.[25] In resource-constrained environments, packet loss and device offline can happen frequently. The structure of systems in such environments is unstable. Therefore, it is reasonable to utilize consensus algorithms to provide robustness for these systems. Suppose there is a cluster C, and there is a consensus algorithm f to help C to agree on some facts. For example, if the members in the cluster need to agree on the fact of who is the leader node, $f(C)$ will be the

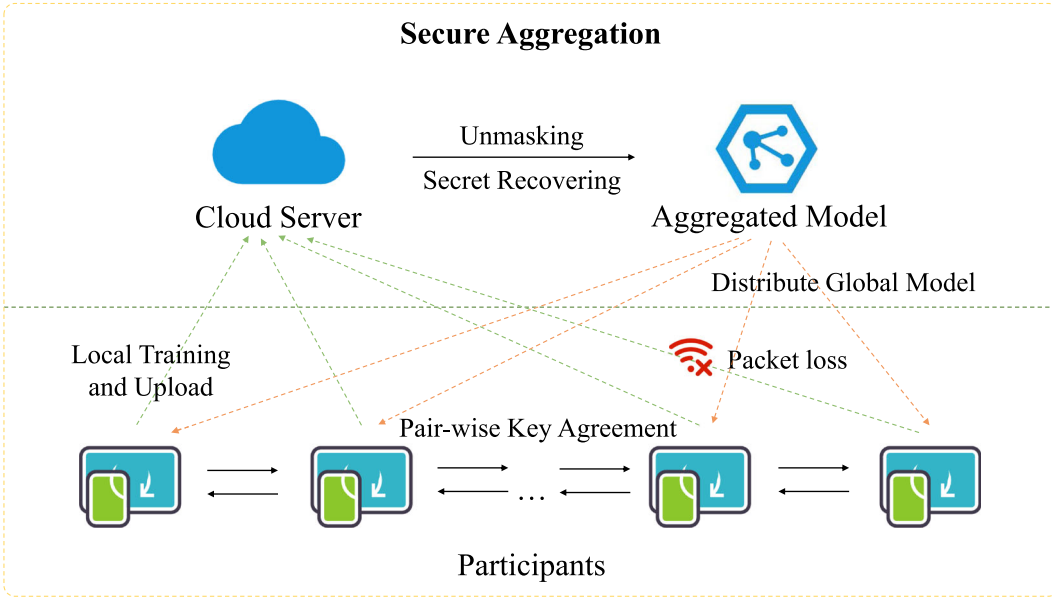

FIGURE 2 The framework of SecAgg[14] framework [Color figure can be viewed at wileyonlinelibrary.com]

answer. In fact, most consensus algorithms let each node compete by canvassing, and by designing different protocols, combined with factors such as the instability of network latency, a node with the most votes is always derived in the end. In this paper, we use a consensus algorithm modified from Raft[18] to maintain the tree structure of participant composition.

3 | METHOD

3.1 | Threat model

In our settings, all the participants in FL are assumed to be *honest-but-curious* (or *semi-honest*).[26] An honest-but-curious participant gathers the information it receives instead of deviating from protocols. The attackers may establish several types of attacks such as inference attack,[4] model extraction attack,[27] and model inversion attack.[28] These attacks exploit the parameters/gradients to realize their objectives. The local parameters have a high probability to be eavesdropped on by the attackers if the communication is not protected.

3.2 | Overview of DemoFL

The proposed algorithm aims to obtain a global model, which is trained with all participants' data without compromising privacy. Figure 3 illustrates the overview of the proposed method. It contains two modules: the tree-structured learning system and the consensus system. In the learning system, DemoFL organizes the learning devices into a tree structure to avoid excessive communication load on a single node. The tree structure also makes it easier to adjust the topology when special issues happen. To protect users' privacy from the attacks mentioned in Section 3.1, the parameters are concealed by MPC-based methods. The learning part contains two phases: tree structure construction and partial-tree-based learning. The tree structure is constructed with the help of a consensus algorithm, and the learning process is recursively carried out in the learning groups. A learning group is a part of the tree structure, which will be introduced in the following subsection. The parameters of the model are merged from the lower to the upper levels of the tree structure during recursive learning. Each learning group uses the secret sharing-based MPC method for secure learning. The consensus system works when communication errors occur in the learning system (e.g., some devices are offline). It will help to reconstruct the tree structure based on the designed consensus algorithm. When a node has left the structure, the consensus algorithm organizes suitable other members to campaign for that missing position and finally arrives at the most suitable node to add to that position. To make it easier to understand our subsequent description, some symbols that will be used are described in Table 1.

3.3 | Tree structure construction

In this subsection, we introduce how the tree structure of the learning system is formed. Before the training process starts, there are N devices that will be the participants of DemoFL. Then these nodes will execute a consensus $\mathbb{C}$ to elect a root. $\mathbb{C}$ is described in Algorithms 1 and 2. Each node has 3 possible identities, that is, follower, candidate, and leader. In one round of execution of $\mathbb{C}$, a follower is a node that has voted for someone else and it will not run for leader. A candidate will

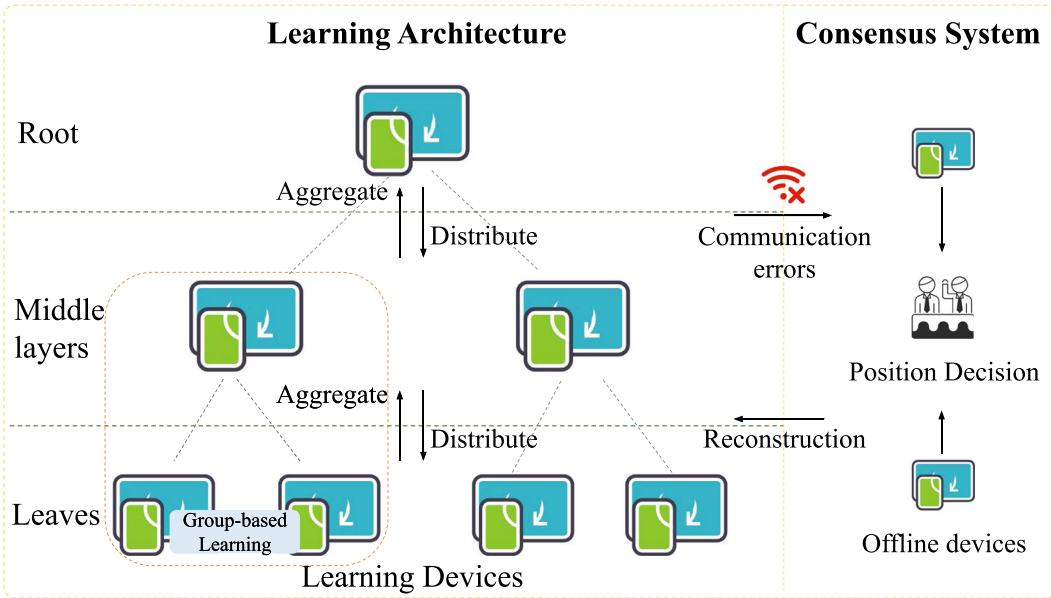

FIGURE 3 The overview of the proposed framework DemoFL. DemoFL, Democratic Federated Learning. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Symbol list

Symbols	Description
N	Total number of nodes in the FL system.
P	The set of all nodes (clients). The ith node is marked as P_i.
S	The root of the tree structure, which will obtain the final model in each round of FL.
$\mathbb{E}$	Total rounds of FL.
$\mathbb{M}$	The maximum number of children nodes that a non-leaf node can have, i.e., the degree of the tree structure.
M_i^j	The trained model possessed by the jth node in the ith epoch. We denote the final obtained model in the ith round by M_i^S.
W_i^j	The parameters of the M_i^j after the training process.
$\mathbb{C}$	A consensus algorithm. It will reconstruct the tree if some non-leaf node in the tree is removed. If $\mathbb{C}$ receives a set of nodes as parameters, it will hold a round of election.

try to gather votes from other followers. Once a candidate has gathered no less than $\frac{\mathbb{N}}{2}$ votes, where $\mathbb{N}$ is the number of nodes that participate in the election, it will consider itself as a leader. Due to some factors such as the value of $\mathbb{N}$, there are situations where more than 1 nodes are in leader state. To address this situation, the candidates will record their communication delays during the vote gathering process. Afterward, the leaders will calculate their average delay for further competition. There are 3 types of messages a node may send to others: "vote", "ask for vote" and "I am leader." The rules for a node to send or process a message are as follows:

1. A follower, after losing connection with the leader, will randomly sleep for a period of time before becoming a candidate and voting for itself.
2. A candidate will broadcast "ask for vote" to all other nodes.
3. A leader will broadcast "I am leader" to all other nodes.
4. A follower sends "vote" to the sender on receiving "ask for vote" and stops trying to become a candidate.
5. A follower or candidate will become a follower on receiving "I am leader" and stops other attempts.
6. On receiving "I am leader", a leader will become a follower if it has fewer votes than the sender. Otherwise, it will remain a leader.
7. When the leader has not been elected at the timeout, the election round starts again.

Algorithm 1: Consensus Algorithm: Root Election

Input: P.
Output: The root node S.
$S \leftarrow -1$;
while $S = -1$ **do**
  for $P_i \in P$ **do**
    $P_i.term \leftarrow 1$;
    $P_i.votes \leftarrow 1$;
    $P_i.delays \leftarrow \emptyset$;
    P_i randomly sleeps for some time ;
    $P_i.state \leftarrow$ "candidate" ;
    `/* Vote for itself.` `*/`
    $P_i.votes \leftarrow P_i.votes + 1$;
    P_i broadcast to other nodes in P to ask for votes ;
  end
  RaftElection() ;
  $V \leftarrow \{P_i \mid P_i.votes \geq \frac{N}{2}\}$;
  if $V \neq \emptyset$ **then**
    Sort V by average delay ;
    $S \leftarrow V[0]$;
    return S ;
  end
  Sleep until all nodes get ready.
end

Algorithm 2: RaftNodeReaction

Data: A node P_i.
if $P_i.votes$ *receives vote from* P_j **then**
  $d_{ij} \leftarrow$ the communication delay ;
  $P_i.delays \leftarrow P_i.delays \cup d_{ij}$;
  $P_i.votes \leftarrow P_i.votes + 1$;
end
if $P_i.votes \geq \frac{N}{\mathbb{M}}$ *and* $leaders < \mathbb{M}$ **then**
  $P_i.state \leftarrow$ "leader" ;
  $leaders \leftarrow leaders + 1$;
  P_i starts broadcasting "I am leader" ;
end
if P_i *receives "ask for vote" from* P_j *and* $P_i.voted < \mathbb{M}$ **then**
  P_i sends 'vote' to P_j ;
  if $leaders >= \mathbb{M}$ **then**
    P_i stops broadcasting ;
  end
end
if P_i *receives "I am leader" and* $P_i.followed < \mathbb{M}$ **then**
  $P_i.followed \leftarrow P_i.followed + 1$;
  if $leaders >= \mathbb{M}$ **then**
    P_i stops broadcasting ;
  end
end

Similar to Raft, a follower can only vote for one node for each term and any node will ignore the messages from a node with a term less than that of its own. When communicating with a node that has a higher term, a node will change its term to the higher one. As depicted in the pseudocode, the root election process is similar to the leader election process in Raft algorithm. The difference is that when there are multiple nodes have gathered enough votes, $\mathbb{C}$ will select the node that has a relatively better network environment as the root instead of re-election. In addition, the election process will not happen periodically because the purpose of the proposed algorithm focuses on robustness and efficiency instead of fairness.

When the root is elected, the other nodes will continue executing $\mathbb{C}$ to find its suitable position in the tree. In this case, these nodes are competing for the children nodes of the current top-level node instead of the root node. The process is depicted in Algorithm 3. It is similar to the root election process. However, there are two major differences. First, a node has $\mathbb{M}$ votes instead of 1, which means a node can support more nodes to win. Second, when a node decides to follow another node, it will not stop until it knows there are already $\mathbb{M}$ leaders. This adjustment accelerates the election process greatly. The number of leaders is transmitted in all communications and is incremented by one when a node becomes a leader. If a larger value is received, the stored value is updated by all nodes. After the vote gathering phase, all the nodes that are in leader state will become children nodes of the corresponding node. Different from root election, the majority is set to $\mathbb{N}/\mathbb{M}$, which means at most $\mathbb{M}$ nodes will win one round of competition. This can reduce the execution times of $\mathbb{C}$, which will be executed repeatedly until there are no remaining nodes. At last, all the FL devices are formed into a tree structure. Although this setup process has significant communication overhead, it is still acceptable considering that it only needs to be established once.

Algorithm 3: Consensus Algorithm: Position Decision

Input: The current tree T, the set of remaining nodes $\mathbb{R}$ and its amount $\mathbb{N}$.
Output: The new tree $T^{`}$.

```
𝕊 ← {ℙ_i | ℙ_i ∈ T ∧ ‖ℙ_i.children‖ < 𝕄} ;
Sort 𝕊 by depth ;
s ← 𝕊[0] ;
for ℙ_i ∈ ℝ do
    ℙ_i.state ← "candidate" ;
    ℙ_i.term ← ℙ_i.term + 1 ;
    ℙ_i.votes ← 1 ;
    ℙ_i.delays ← ∅ ;
    ℙ_i randomly sleeps for some time ;
    ℙ_i broadcast to other nodes;
end
RaftElection() ;
V ← {ℙ_i | ℙ_i.votes ≥ ℕ/𝕄} ;
if V ≠ ∅ then
    for ℙ_i ∈ V do
        if ‖s.children‖ < 𝕄 then
            s.add_children(ℙ_i) ;
        end
    end
end
```

3.4 | Tree-based secure learning

In this subsection, we introduce how the learning is conducted based on the constructed tree structure. The learning process is from bottom to top, that is, it starts with the leaf nodes first.

The tree is divided into many learning groups. Each non-leaf node with all its direct children nodes forms a learning group. Note that a non-leaf and non-root node can appear in two learning groups, where they perform the parent and the child respectively.

3.4.1 | Learning group

We take one learning group as an example to explain the learning process. Suppose there is a learning group $\mathbb{G}$ with n nodes in it. In each learning round, this group will receive a model M^S from the root. Then each node will train this model with its private data and obtain the parameters respectively. Finally, these parameters will be aggregated and the aggregated model will be acquired. This process can be expressed as

$$M_{i+1}^S = M_i^S + \frac{\sum_{j=1}^{n} C_j W_i^j}{C}, \tag{1}$$

where C is the total number of training samples used in this group and C_j is the number of the jth node. Though some works protect C_i from semi-honest attackers,[29] leakage of C_i does not help attackers to reconstruct any data. In addition, C can also be aggregated based on multi-party addition since it only needs addition to compute C from all the components C_i, $i \in [1, n]$. Therefore, the nodes in a group will use the secret sharing algorithm to get C before the learning starts. Now the FedAvg is converted to a form where only secure addition is needed. Different from SecAgg, a pair-wise key agreement is not established in a learning group. The root of a learning group and its first child will be the aggregation centers. Therefore, the members of a learning group will split their secret into two pieces instead of n. This improves the efficiency greatly and makes recovering secrets using the t-out-of-n mechanism hardly possible to achieve and reduces some privacy. However, in resource-constrained scenarios, efficiency is more important than privacy. For each member of a learning group to know who the aggregation center is, they are informed when they join that learning group. Then the steps of secure aggregation in $\mathbb{G}$ are described as follows:

1. Each node in $\mathbb{G}$ trains M_i^S with corresponding data and obtains parameters W_i^j.
2. $\mathbb{G}_j$ splits $C_j W_i^j$ into two pieces and sends them to the aggregation centers in $\mathbb{G}$ (the root and its first child node).
3. After aggregation centers has received all pieces from other nodes, they calculate the sum of these pieces.
4. The root's first child sends its sum to its parent node and the average parameter is computed.

Before the learning process starts, C is also shared in the same way. Through the above-described method, a learning group can privately conduct a round of FL.

3.4.2 | Secure aggregation in the partial tree structure

The learning process is performed in a form similar to post-order traversal. When an FL round starts, S distributes the current model to its children, who also recursively send the model forward. When the model is distributed to all leaf nodes, the traversal process starts

backtracking and the non-leaf nodes at the lowest level will start the learning process in their learning group. When a learning group has finished learning, it will send the aggregated model to the parent node. At last, S will gather all the sums of local parameters, and the model can be updated. A clearer description of this process can be referred to Algorithm 4.

Algorithm 4: RecAgg

Input: Current node s, global model M^S.
$s.M \leftarrow M^S$;
if z *is leaf node* **then**
  return ;
else
  for $z \in s.children$ **do**
    $RecAgg(z)$;
  end
  $\hat{W} \leftarrow Add(s.W|s.children.W)$;
  if $s \neq S$ **then**
    Send $\hat{W}$ to $s.parent$;
  else
    $\overline{W} \leftarrow \sum\{\hat{W}\}$;
    $\overline{C} \leftarrow \sum\{\hat{C}\}$;
    $\overline{W} \leftarrow FedAvg(\hat{W}, \hat{C})$;
    $M^S \leftarrow M^S.update(\overline{W})$;
  end
end

Note that the summation process of C is not presented in the pseudocode, since it is very similar to the parameter aggregation procedure. On the other hand, to increase the efficiency, C is not summed separately in each learning group, but only once in total. That means that a learning group will send the sum of local parameters to the parent node instead of average parameters. After the sub-tree learning process, the root node will obtain an aggregated parameter $W_i^S = \sum \frac{C_i W_i^j}{C}$ as the FedAvg.[1]

3.5 | Tree structure reconstruction

To date, we have introduced a tree-structured FL framework. The advantages of this framework are demonstrated when its participants drop out. There is a timer $\mathbb{T}$ in DemoFL, which is used to perceive a drop out. When a node waits for a message from another node for more than $\mathbb{T}$, it determines that the node has dropped out, and tries to start a corresponding reconstruction process with other associated nodes. Depending on the type of node that is dropped, different countermeasures are used. The dropped nodes are categorized into three types: leaf node, non-leaf node, and root. The countermeasures are illustrated as follows:

- *Leaf node*: The drop of a leaf node does not cause a change in the tree structure. Therefore, no measures will be taken.
- *Non-leaf node*: In this case, the structure of the tree inevitably has to change. Suppose a non-leaf node a has dropped out. When its child nodes find that, they will start to compete for leader by executing the root election phase of $\mathbb{C}$. Denote the winner by b. b then send "member change" message to the parent node of a and a will add b to its children set. As for the original children of b, they do not execute $\mathbb{C}$ to get new positions. To improve the efficiency, the adjustment of the structure is designed recursive. b will randomly select one of

its children. Suppose the selected node by t. t and the original children of a become the new children of b. The other children of b will repeat what b did until the algorithm reaches a leaf node.

- *Root*: When the root dropped out, the key information is lost and the whole structure needs to be reconstructed by executing $\mathbb{C}$. However, the lasted model is stored in all nodes therefore the newly elected root can continue the learning process without interruption.

To accomplish the above process, the nodes in the learning tree are required to store some information. First, each node should know its parent node and the root node and have the capability to communicate with them. Second, the root should have knowledge about the whole tree. When a node has won a competition for the new parent, it will ask the root for where the original parent's position is.

3.5.1 | Structure cache

In some situations, some nodes are only temporarily disconnected, and it will quickly rejoin the cluster. However, the node's position may has been replaced by its children. Frequent changes in the tree structure are not effective for FL, therefore we introduce a structure cache mechanism. Structure cache requires each node to record the previous parent node when its position will change. When a non-root node is disconnected, its parent also record its position. With these information, when a node rejoin the cluster, it can quickly find its original parent and position. At this time, all its original children will come back to its sub-tree. Note that a node only records the most recent previous parent node. This means that if any of the nodes involved has experienced no less than two structure changes, the node that dropped the first time cannot be restored to its original position through the cache. This also indicates structure cache can handle the unstable network condition, for example, intermittent network connection.

4 | EFFICIENCY ANALYSIS

First, we analyze the set-up process of DemoFL. Consider an execution of $\mathbb{C}$ with $\mathbb{N}$ nodes engaged in. Each candidate node will broadcast "ask for vote" message to all $\mathbb{N}-1$ nodes and a leader will broadcast "I am leader". At last, any follower has sent an "vote" message to others. In the proposed consensus algorithm, $\mathbb{M}$ winner will be selected in one round, which gets $\frac{\mathbb{N}}{\mathbb{M}}$ votes from others. This indicates that the expectation of candidates is also $\mathbb{M}$. Therefore, the number of communications can be considered as $O(\mathbb{M}\mathbb{N})$ because a winner will broadcast its message to all other nodes. This process may be repeated several times when a round of elections is failed, but empirically, this number of repetitions is not large and can be considered constant. Denote the retry times by t. In addition to the root node election, the other nodes will also run several rounds to select their positions. The number of rounds is expected to be executed as $\frac{N-1}{\mathbb{M}}$, where $\mathbb{M}$ is the degree of the tree. Therefore, the total number of communications can be computed as follow:

$$t \sum_{N>i\times\mathbb{M}} O(\mathbb{M}(N - i \times \mathbb{M})) = O\left(t\left(N + \frac{N^2}{\mathbb{M}}\right)\right). \tag{2}$$

Therefore, the average time complexity of each node in the set-up is $O\left(t\frac{N}{\mathbb{M}}\right)$, which is acceptable for FL frameworks. Then we analyze the sub-tree learning process. In traditional FL frameworks, the server node will send the model to all other nodes, which causes $O(N)$ communication overhead for the server node. In DemoFL, the model is distributed through a tree structure. It means that each non-leaf node will send the global model to $\mathbb{M}$ nodes. In other words, DemoFL splits the burden of one node into multiple nodes. After the model is distributed, each node in a learning group will send a piece of parameters to all other nodes within the same group when it has finished local training. Therefore, the number of communications of each leaf node is usually $\mathbb{M}$. Since each non-leaf node (except the root) belongs to two learning group, their average communication number is approximately $2\mathbb{M}$. Thus, the time complexity of learning for each node is $O(\mathbb{M})$, which is less than $O(N)$ in traditional FL. Therefore, the average complexity of a node is reduced $M = \frac{N}{\mathbb{M}}$ times. In practice, the learning process is usually simultaneous for each learning group. After learning, a learning group will upload its gradients to its parent node. In addition, with smaller $\mathbb{M}$, more learning groups will be formed, which may cause frequent structure reconstruction. Therefore, it is necessary to take a reasonable value for $\mathbb{M}$. Consider optimizing $O(\mathbb{M}) + O(\frac{N}{\mathbb{M}})$, which means the communication complexity of a single learning group and the total communication of aggregation respectively in one round. The best value of $\mathbb{M}$ is $\sqrt{N}$ because $\mathbb{M} + \frac{N}{\mathbb{M}} \geq 2\sqrt{N}$. In fact, both complexities have uncertain constants. Therefore, the actual desirable value of $\mathbb{M}$ is not definitely $\sqrt{N}$, but it is not far different from it.

Finally, we analyze the overheads of reconstruction. We only consider the situation where a non-leaf node is dropped out because a leaf node does not trigger the reconstruction and the root will totally restart the algorithm. As described in Section 3.5, the reconstruction process will start with the execution of $\mathbb{C}$ within a learning group, whose time complexity is $O(\mathbb{M})$. Afterward, the corresponding node will recurse to the leaf node. The time complexity is $O(d\mathbb{M})$, where d is the depth of the tree. Therefore, the communication overhead is $O(d\mathbb{M} + \mathbb{M}) = O(d\mathbb{M})$.

5 | EXPERIMENTAL RESULTS

5.1 | Implementation

The experiments are conducted on a commodity machine running Centos 7 with Intel(R) Xeon (R) Silver 4214R CPU and 256GB memory. Our framework is implemented with Pytorch. We use MNIST and CIFAR-10 as datasets, which are also used in FedAvg.[1] All models are trained in a single thread without distributed acceleration to facilitate comparing and analysis. The optimizer was stochastic gradient descent (SGD) and the learning rate is 0.01 when instructions are absent.

5.2 | Accuracy

Since our work does not modify the learning module compared to other FL frameworks, it should not influence the accuracy of models. We conduct a series of experiments to verify this fact. Since this experiment aims to prove validity instead of high accuracy, the models were not

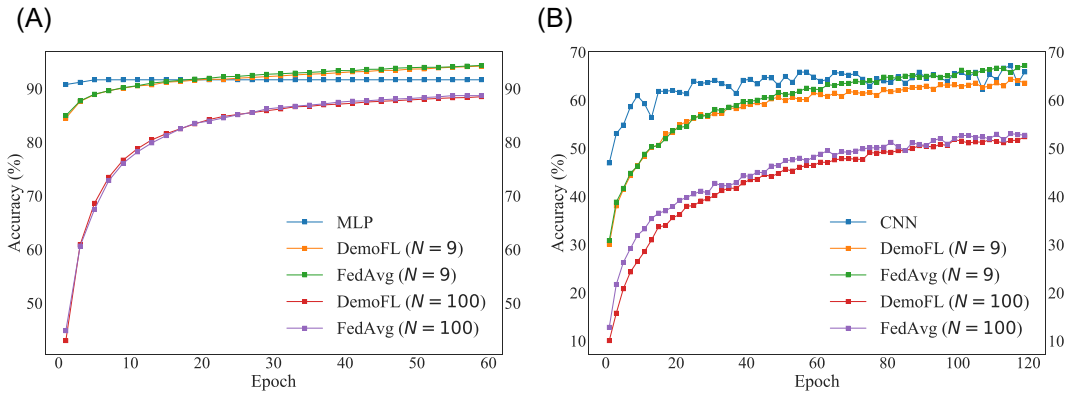

FIGURE 4 The accuracy comparison of DemoFL with FedAvg and baselines. (A) Results on MNIST and (B) results on CIFAR-10. DemoFL, Democratic Federated Learning. [Color figure can be viewed at wileyonlinelibrary.com]

optimized on some parameters such as learning rate. We conducted experiments on datasets MNIST and CIFAR-10. We have two settings for evaluation. The first has nine nodes and $\mathbb{M}$ is set to 3. This indicates that there are three learning groups in the tree structure, with the number of learning members 4, 4, and 3. The second has 100 nodes and $\mathbb{M}$ is set to 10. These two settings evaluate DemoFL on a small and large number of nodes respectively. Assuming no dropouts or quits occur during training. The training data set is distributed to all nodes with an average size in DemoFL. MNIST is evaluated with a two-layer MLP and we use a CNN model for evaluation on data set CIFAR-10, which is three-layer with two convolutional layers. Figure 4 shows the result. The results show that DemoFL and the normal model converge to the same range. Except for the early stages of training, the final models have almost the same effects on the test data set. When the number of clients are large, the model will converge slower and needs more rounds to perform as well as other models.

5.3 | Time overhead

DemoFL reduces the communication complexity for a node in a training epoch from $O(N)$ to $O(\mathbb{M})$. However, in the set-up phase, the tree construction process selects a root from the members and organizes the other nodes. This process may lead to extra communication overheads. We compare the efficiency of the tree construction process with the set-up process of SecAgg.[14] In SecAgg, the factor that can have an impact on efficiency is the number of participants. In DemoFL, the factors that affect the efficiency include the maximum number of children a node has in addition to the number of participants. We evaluate the efficiency performance with two metrics, the average number of communications for a single node and the time cost of the establishment phase. The communications in the experiments are conducted in a LAN environment.

The results are illustrated in Figure 5A. It is observed that the number of communications and time cost of DemoFL increases close to linearly with the number of clients. The volatility is caused by the different number of rounds executed in each round of the consensus algorithm, versus the number of communications that need to be made in each round. When the number of clients increases, the consensus algorithm may

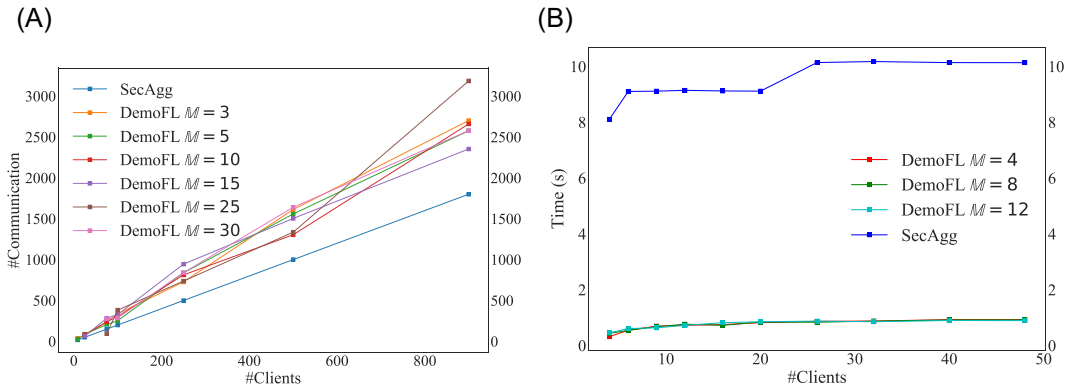

FIGURE 5 The efficiency comparison of DemoFL with SecAgg in set-up process. (A) Average number of communications per client and (B) average time cost per client. DemoFL, Democratic Federated Learning. [Color figure can be viewed at wileyonlinelibrary.com]

fail several times, which leads to more communications. In SecAgg, the number of communications a node needs to make is fixed.

On the time spent, the average time spent by a single client in SecAgg and DemoFL is almost constant. This is because all nodes send information to others in parallel, therefore the contribution to the time spent is the communication spent per node. Since nodes in SecAgg require more cryptographic calculations such as generating secret keys and random numbers, their average time consumption is a bit higher. Note that the time spent shown in Figure 5B means the average communication time spent on each node. It is apparent that both time expenditures are negligible for the overhead of machine learning model training.

5.4 | Robustness

We consider two kinds of communication errors. First is the packet loss, which means the local parameters of a node will not be received in group learning process. In such cases, DemoFL will ignore this local parameter and its sample size coefficient. While in SecAgg, the parameters will be restored based on t-out-of-n mechanism. This difference may lead to an impact on the final accuracy of the model. We conduct sufficient experiments to trace the impact. Suppose there DO is the proportion of nodes that will lose packets in each round of aggregation. $\mathbb{M}$ is set to 9 and N is set to 20. Since SecAgg will restore the lost parameters, we only compare our framework with nonfederated models. The results in Figure 6A indicates that when the packet loss rate is within a reasonable range, it doesn't impact the performance of the model. When the packet loss rate is quite high (15%), it slows down the performance of the model. The second communication error is long-term offline, which will change the topology of the tree structure. This error may cause extra communication overheads. Since inserting new nodes into the tree structure does not need a consensus algorithm, we do not consider the situation where new nodes come to participate. We statistically measure the additional communication that would be caused by a node drop at different $\mathbb{M}$ and different heights. The experimental results are shown in Figure 6B. The additional communications increase nearly linearly with both the depth and the degree of the tree. Apparently, the overhead is within acceptable limits and the average communication cost of nodes is barely elevated.

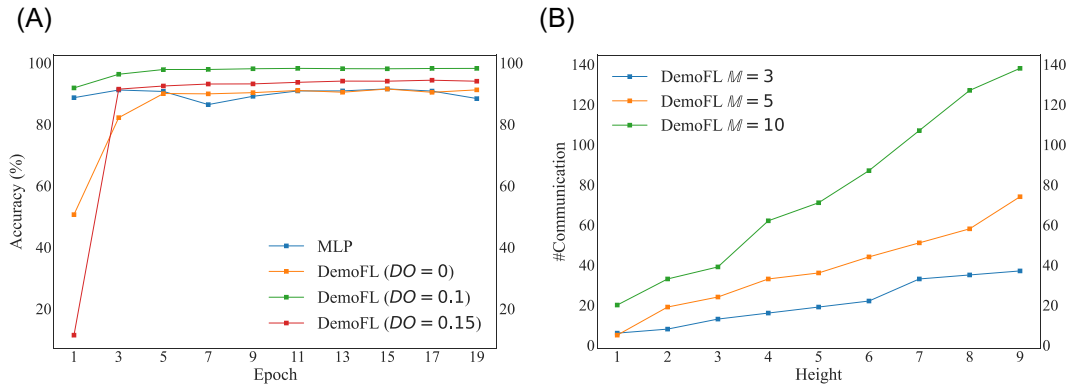

FIGURE 6 The tolerance of communication errors and new members of DemoFL. (A) Accuracy on different dropout rates and (B) the extra communication overhead caused by one node droppingat different heights. The height is counted from the leaf. DemoFL, Democratic Federated Learning. [Color figure can be viewed at wileyonlinelibrary.com]

6 | RELATED WORK

Homomorphic encryption (HE)-based solutions are intuitively effective to solve aggregation problems, and there are many researchers tried to reduce the overhead caused by HE. Hardy et al.[30] designed an additively homomorphic encryption method for FL in 2017. However, the encryption method is too complicated that it does not reach high efficiency. Ma et al. proposed xMK-CKKS,[31] which is designed for IoT devices. However, it is computationally time-consuming, and HE-based methods are not suitable for most resource-constrained devices. The state-of-the-art HE method was proposed by Zhang et al.[32] It reduced the encryption overhead greatly at the cost of trivial loss of accuracy. However, it still costs much time on computation compared to MPC-based and DP-based methods.

DP-based frameworks add noises to the parameters to deceive the attackers. The problem is that these noises also have an influence on the learning result.[33,34] Therefore, there remains a trade-off problem between the performance and privacy levels for DP methods in FL. Moreover, the effects of DP-based methods are barely satisfactory.[13] Geyer et al.[35] tested DP in FL in 2017 and the result showed that DP's influence on accuracy is nontrivial. Following their work, Bayesian differential privacy[36] was proposed in 2019. Bayesian differential privacy considered the probability and distribution of data and is effective for machine learning models whose data are often restricted to a particular type. While it may be inefficient in vertical FL situations. Moreover, all these DP-based methods did not prove DP does not impact the accuracy in other more complicated and large-scale models, which means it is different to find a universal DP method for all machine learning models. Additionally, Jayaraman et al.[13] indicated that the guarantees of DP are essentially meaningless and DP implementations may provide unacceptable utility-privacy trade-offs in practice.

MPC-based methods have the least computation cost but require more communication. Bonawitz et al.[14] utilized Diffie-Hellman key exchange to generate pairwise secret masks, based on which MPC protocol is enabled. However, its high communication overhead is not suitable in low-power situations. A recent work by Kanagavelu et al.[15] also analyzed the MPC process in FL and tried to reduce the overheads by introducing a scheme named "aggregation committee," yet this method does not take the robustness into consideration.

In addition, blockchain-based methods[37–39] are also promising and can provide a high-security level, yet the overheads of blockchain-based methods are too heavy for the end devices in resource-constrained environments such as LPWAN.

7 | CONCLUSION AND FUTURE WORKS

In this paper, we propose a communication-efficient, and privacy-preserving FL framework named DemoFL. It adopts MPC-based secret sharing methods to achieve privacy-preserving against honest-but-curious adversaries and maintains the topology of participants with the help of designed consensus algorithms. Our experimental results indicate that DemoFL has a quite cheap time complexity on either the set-up process or the training process, and it has a high resistance to communication errors. These features enable DemoFL to be established in resource-constrained environments. There has been an increasing demand for the efficient use of FL methods to build models in various scenarios. Therefore, exploring how to significantly reduce the communication overhead of FL algorithms for a variety of devices while protecting privacy could be future research.

ACKNOWLEDGMENTS

This study was supported by the National Key R&D Program of China (Grant No.2020AAA0107703), the National Natural Science Foundation of China (Grant Nos. 62076125, U20B2049, U20B2050).

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Zhaoyang Han http://orcid.org/0000-0002-1511-361X
Lu Zhou http://orcid.org/0000-0001-6240-6688
Chunpeng Ge http://orcid.org/0000-0003-2629-7220
Zhe Liu http://orcid.org/0000-0001-8578-2635

REFERENCES

1. McMahan B, Moore E, Ramage D, Hampson S, Arcas BAy. Communication-efficient learning of deep networks from decentralized data. In: Singh A, Zhu XJ, eds. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*. PMLR; 2017:1273-1282. http://proceedings.mlr.press/v54/mcmahan17a.html
2. Yang T, Andrew G, Eichner H, et al. Applied federated learning: Improving google keyboard query suggestions. 2018. https://arxiv.org/abs/1812.02903
3. Xiao Z, Xu X, Xing H, Song F, Wang X, Zhao B. A federated learning system with enhanced feature extraction for human activity recognition. *Knowl-Based Syst*. 2021;229:107338.
4. Shokri R, Stronati M, Song C, Shmatikov V. Membership inference attacks against machine learning models. 2017 IEEE Symposium on Security and Privacy (SP). 2017:3-18.
5. Wang Z, Song M, Zhang Z, Song Y, Wang Q, Qi H. Beyond inferring class representatives: User-level privacy leakage from federated learning. IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE; 2019: 2512-2520

6. Li Z, Huang Z, Chen C, Hong C. Quantification of the leakage in federated learning. arXiv preprint arXiv:1910.05467. 2019.
7. Nasr M, Shokri R, Houmansadr A. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. 2019 IEEE Symposium on Security and Privacy (SP); 2019:739-753.
8. Shi E, Chan TH, Rieffel E, Chow R, Song D. Privacy-preserving aggregation of time-series data. Proc. NDSS. Vol 2. Citeseer; 2011:1-17.
9. Pillutla K, Kakade SM, Harchaoui Z. Robust aggregation for federated learning. *Trans Signal Process*. 2022;70:1142-1154. doi:10.1109/TSP.2022.3153135
10. Bonawitz KA, Salehi F, Konečný J, McMahan B, Gruteser M. Federated learning with autotuned communication-efficient secure aggregation. In: Matthews MB. *53rd Asilomar Conference on Signals, Systems, and Computers, ACSCC 2019, Pacific Grove, CA, USA, November 3-6, 2019*. IEEE; 2019:1222-1226. doi:10.1109/IEEECONF44664.2019.9049066
11. Mandal K, Gong G. PrivFL: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks. Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW'19, New York, NY, USA. Association for Computing Machinery; 2019:57-68
12. Acar A, Aksu H, Uluagac AS, Conti M. A survey on homomorphic encryption schemes: theory and implementation. *ACM Comput Surv* 2018;51(4):1-35.
13. Jayaraman B, Evans D. Evaluating differentially private machine learning in practice. 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA. USENIX Association; 2019:1895-1912. https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman
14. Bonawitz K, Ivanov V, Kreuter B, et al. Practical secure aggregation for privacy-preserving machine learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security CCS'17, New York, NY, USA. Association for Computing Machinery; 2017:1175-1191. doi:10.1145/3133956.3133982
15. Kanagavelu R, Li Z, Samsudin J, et al. Two-phase multi-party computation enabled privacy-preserving federated learning. 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020, Melbourne, Australia, May 11-14, 2020. IEEE; 2020:410-419.
16. Zhu H, Goh RSM, Ng WK. Privacy-preserving weighted federated learning within the secret sharing framework. *IEEE Access*. 2020;8:198275-198284.
17. Mekki K, Bajic E, Chaxel F, Meyer F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express*. 2019;5(1):1-7.
18. Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA. USENIX Association; 2014:305-319. https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro
19. Yang Q, Liu Y, Chen T, Tong Y. Federated machine learning: concept and applications. *Trans Intell Syst Technol*. 2019;10(2):1-19. doi:10.1145/3298981
20. Mohassel P, Rosulek M, Zhang Y. Fast and secure three-party computation: The garbled circuit approach. Proceedings of the 22nd ACM SIGSAC Conference on Computerand Communications Security, CCS'15, New York, NY, USA. Association for Computing Machinery; 2015:591-602. doi:10.1145/2810103
21. Shamir A. How to share a secret. *Commun ACM*. 1979;22(11):612-613.
22. Rabin MO. How to exchange secrets with oblivious transfer. Harvard University Technical Report 81 talr@watson.ibm.com 12955, received 21 Jun 2005. 2005. http://eprint.iacr.org/2005/187
23. Beimel A. Secret-sharing schemes: a survey. In: Chee YM, Guo Z, Ling S, Shao F, Tang Y, Wang H, Xing C, eds. *Coding and Cryptology*. Springer Berlin Heidelberg; 2011:11-46.
24. Nguyen G, Kim K. A survey about consensus algorithms used in blockchain. *J Inf Process Syst*. 2018;14(1):101-128.
25. Weil SA, Brandt SA, Miller EL, Long DDE, Maltzahn C. Ceph: A scalable, high-performance distributed file system. *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI'06, USA. USENIX Association; 2006: 307-320.
26. Brickell J, Shmatikov V. Privacy-preserving graph algorithms in the semi-honest model. In: Roy B, ed. *Advances in Cryptology—ASIACRYPT 2005*. Springer Berlin Heidelberg; 2005:236-252.

27. Tramèr F, Zhang F, Juels A, Reiter MK, Ristenpart T. Stealing machine learning models via prediction APIs. Proceedings of the 25th USENIX Conference on Security Symposium, SEC'16, USA. USENIX Association; 2016:601-618.
28. Fredrikson M, Jha S, Ristenpart T. Model inversion attacks that exploit confidence information and basic countermeasures. Proceedings of the 22nd ACM SIGSAC Conference onComputer and Communications Security, CCS'15, New York, NY, USA. Association for Computing Machinery; 2015:1322-1333. doi:10.1145/2810103.2813677
29. Zhu H, Li Z, Cheah M, Goh RSM. Privacy-preserving weighted federated learning within oracle-aided MPC framework. *CoRR*. 2020. https://arxiv.org/abs/2003.07630
30. Hardy S, Henecka W, Ivey-Law H, et al. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *CoRR*. 2017; abs/1711.10677.
31. Ma J, Naas S, Sigg S, Lyu X. Privacy-preserving federated learning based on multi-key homomorphic encryption. *CoRR*. 2021; abs/2104.06824.
32. Zhang C, Li S, Xia J, Wang W, Yan F, Liu Y. BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning. 2020 USENIX Annual Technical Conference (USENIX ATC 20). USENIX Association; 2020:493-506. https://www.usenix.org/conference/atc20/presentation/zhang-chengliang
33. Abadi M, Chu A, Goodfellow I, et al. Deep learning with differential privacy. Proceedingsof the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS'16, New York, NY, USA. Association for Computing Machinery; 2016:308-318. doi:10.1145/2976749.2978318
34. Wei K, Li J, Ding M, et al. Federated learning with differential privacy: algorithms and performance analysis. *IEEE Trans Inf Forensics Secur*. 2020;15:3454-3469.
35. Geyer RC, Klein T, Nabi M. Differentially private federated learning: a client level perspective. *CoRR*. 2017. http://arxiv.org/abs/1712.07557
36. Triastcyn A, Faltings B. Federated learning with bayesian differential privacy. 2019 IEEE International Conference on Big Data (Big Data); 2019:2587-2596.
37. Weng J, Weng J, Zhang J, Li M, Zhang Y, Luo W. DeepChain: auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Trans Depend Secure Comput*. 2019;18(5):2438-2455.
38. Lu Y, Huang X, Dai Y, Maharjan S, Zhang Y. Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Trans Ind Inform*. 2020;16(6):4177-4186.
39. Kim H, Park J, Bennis M, Kim S. Blockchained on-device federated learning. *IEEE Commun Lett*. 2020;24(6):1279-1283.

How to cite this article: Han Z, Zhou L, Ge C, Li J, Liu Z. Robust privacy-preserving federated learning framework for IoT devices. *Int J Intell Syst*. 2022;37:9655-9673. doi:10.1002/int.22993