

PRESENTED AT THE ISCISC'2024 IN TEHRAN, IRAN.

## Shapley Value for Federated Learning: A Distributed and Fair Framework<sup>☆</sup>

Mohammad Amin Sarzaem<sup>1</sup>, Seyed Reza Hoseini Najarkolaei<sup>1,\*</sup>, and  
Mohammad Reza Aref<sup>2</sup>

<sup>1</sup>Information Systems and Security Lab. (ISSL), Sharif University of Technology, Tehran, Iran

<sup>2</sup>Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran

### ARTICLE INFO.

#### Keywords:

Federated Learning, Shapley Value, Distributed Coded Computing, Polynomial Codes

#### Type:

Research Article

#### doi:

10.22042/isecure.2025.219572

### ABSTRACT

In a federated learning system, the objective is to train a global model over distributed datasets without centralizing all data on a single unit. This is accomplished by training a local model on the dataset of each data owner and then aggregating these local models to preserve the datasets' privacy. To incentivize clients to actively engage in the learning process, fairness-aware federated learning techniques can be employed. One such approach involves quantifying the contribution of locally trained models in training the global model by Shapley value (SV) using an additional dataset and rewarding them according to their contributions. However, the calculation of the Shapley value presents a significant challenge due to its high computational complexity. To tackle this issue, our research presents a contribution-based federated learning method that efficiently computes the contribution of each locally trained model by distributing the additional dataset among processing nodes in a private manner and calculating the Shapley value over them.

© 2025 ISC. All rights reserved.

## 1 Introduction

The past decades have witnessed a growth in high-powered devices like mobile phones, generating vast amounts of data across various platforms. There is a growing interest in processing large datasets for applications like behavior recognition [1], next-word recommendation [2], and face detection [3], utilizing

machine learning algorithms to extract knowledge from the data. Decentralized data sources have led to the development of federated learning. This approach distributes data and computation across different machines, enabling the training of models without the need for extensive data collection or storage. In this setting, some data owners possess private datasets, and the objective is to train a model across all data without exposing any private information.

In a federated learning system, each data owner trains a local model on their private dataset and sends it to a central master node. The master node aggregates the locally trained models, performs computations, and broadcasts the updated global model

<sup>☆</sup> The ISCISC'2024 program committee effort is highly acknowledged for reviewing this paper.

\* Corresponding author.

Email addresses: [masz.netcom@hotmail.com](mailto:masz.netcom@hotmail.com),

[hoseini.education@gmail.com](mailto:hoseini.education@gmail.com), [aref@sharif.edu](mailto:aref@sharif.edu)

ISSN: 2008-2045 © 2025 ISC. All rights reserved.

back to each data owner for the next training iteration. Federated learning has been applied in various domains [4], such as training next-word prediction algorithms collaboratively on mobile phones [2] and extracting statistical insights from sensitive organizational data—for example, using hospital patient records for privacy-preserving medical data analysis [5]. Additionally, federated learning is utilized in IoT systems [6], where real-time model updates are essential, such as in autonomous vehicles. Across all these applications, the primary focus remains on maintaining data privacy while effectively training models and extracting valuable insights from distributed datasets.

Despite its advantages, federated learning faces several challenges, including communication complexity, unbalanced system capacity, statistical heterogeneity, and privacy concerns [4]. While data owners do not share their raw private data, studies such as [7] have demonstrated that the master node can infer certain information through model inversion attacks, leading to potential privacy breaches. Some aggregation algorithms use secure multi-party computation (SMC) to provide secure aggregation of local models, such as lightSecAgg [8]. Additionally, differential privacy is a common approach for preserving the privacy of local datasets, such as [9], [10], which guarantees privacy at the cost of reducing efficiency. In differentially private FL, data owners add noise to their local models and send them back to the master. While this approach keeps the models private, it may decrease the convergence rate of the algorithm and reduce the precision of the final model. Other approaches, such as [11], provide privacy-preserving byzantine-tolerant federated learning using coded computing.

One of the other challenges of FL is reliability, which refers to how we can trust data owners to implement the correct algorithm on their local data and send the accurate model to the master. To address this issue, fairness-aware federated learning approaches have been proposed. A survey on this topic can be found in [12]. In [13], a contribution-based FL system is proposed, in which the master incentivizes the data owners based on their contribution to the final model and they receive a reward based on it. Moreover, more weight can be given to the local model with more contribution in updating the global model at each training round. Some of the contribution-based FL systems use the Shapley value for computing contribution, which is a fair method in coalitional game theory to compute each player's contribution to the total payoff [14].

As we will see in Section 3, computing the Shapley value requires evaluating a payoff function for each

permutation of the players; hence, it demands significant computational power for all permutations of data owners. Additionally, evaluating contributions in a federated learning system may use a large labeled dataset, resulting in substantial computational demands. Both factors slow down the model contribution evaluation step. In [15], it is shown that computing the Shapley value may take up to 99 percent of the total computations in a federated learning system. To tackle this challenge, the most common approach has been to use the Monte Carlo sampling theorem [14]. Furthermore, [15] proposes a non-uniform sampling method to improve sampling performance. However, even with these methods, the contribution evaluation step remains time-consuming.

Many efforts have been conducted recently to make Shapley value computation more efficient in federated learning. [16] attempts to apply the Shapley value to create a fair, decentralized federated learning system based on blockchain efficiently. [17] mentions several estimation methods to make Shapley value computations more affordable and also proposes a new one. Additionally, it discusses future directions such as fast estimation, applications of the Shapley value beyond federated learning, and applying privacy constraints to local models using secure multi-party computation. In fact, the assessment of contributions requires knowledge of local models, which may violate the privacy of data owners' local models.

In this paper, we introduce a new contribution-based federated learning system based on distributed coded computing to address the computational challenges of previous systems. In this system, data owners have common features but possess different instances of data. We compute each data owner's contribution using the Shapley value, and data owners may receive payment or rewards based on their contributions. The contribution evaluation process is carried out in a distributed manner to enhance system speed. It is important to note that we distribute the evaluation dataset in a private manner to prevent data owners from accessing and potentially exploiting it to increase their contributions.

To preserve the privacy of the evaluation dataset, we use Lagrange coded computing [18]. We demonstrate that using LCC enables efficient Shapley value computation while significantly reducing evaluation time in terms of the size of the evaluation dataset. Furthermore, we discuss how secure multi-party computation can ensure the privacy of local models during Shapley value computations.

The remainder of this paper is structured as follows: Section 2 introduces relevant preliminaries, including the Shapley value. Section 3 outlines the problem

setting, and Section 4 details our proposed scheme. Finally, in Section 5, we evaluate the computational complexity of our approach and compare it to existing methods.

## 2 Preliminaries

Before describing our proposed scheme, we need to introduce some preliminaries.

### 2.1 Shapley Value (SV)

First, we introduce Shapley value. Let  $\mathcal{N} = \{1, \dots, n\}$  be a finite set of players. Each subset of  $\mathcal{N}$  is called a coalition, and the set  $\mathcal{N}$  is called the grand coalition. A coalitional game is defined by the pair  $(\mathcal{N}, v)$ , where  $v : 2^{\mathcal{N}} \Rightarrow \mathbb{R}$  is a payoff function relating each coalition to its payoff, and  $v(\emptyset) = 0$ . We need to define a contribution value  $\phi_i$  for each player  $i$ , ensuring a fair division of the total payoff among players. This value is defined based on some axioms.

First, the sum of all players' contributions must equal the grand coalition's payoff ( $\sum_i \phi_i = v(\mathcal{N})$ ). Second, for a dummy player  $i$  that  $v(\mathcal{S} \cup i) = v(\mathcal{S}), \forall \mathcal{S} \subseteq \mathcal{N} \setminus i$ , its contribution equals zero ( $\phi_i = 0$ ). Third, if two players  $i$  and  $j$  have the same marginal contributions across all coalitions, i.e.  $v(\mathcal{S} \cup i) = v(\mathcal{S} \cup j), \forall \mathcal{S} \subseteq \mathcal{N} \setminus \{i, j\}$ , their contributions must be the same. The last axiom is linearity which means if we have two coalitional games  $(\mathcal{N}, v)$  and  $(\mathcal{N}, w)$ , each player's contribution in the game  $(\mathcal{N}, v + w)$  must be the sum of that player's contributions in each game ( $\phi_i(\mathcal{N}, v + w) = \phi_i(\mathcal{N}, v) + \phi_i(\mathcal{N}, w), \forall i \in \mathcal{N}$ ).

For any coalitional game  $(\mathcal{N}, v)$ , Shapley value for each player  $i$  is given by:

$$\phi_i = \frac{1}{|\Pi(\mathcal{N})|} \sum_{\pi \in \Pi} [v(\mathcal{P}_{\pi}^i \cup \{i\}) - v(\mathcal{P}_{\pi}^i)] \quad (1)$$

where  $\Pi(\mathcal{N})$  is the set of all permutations of  $\mathcal{N}$ ,  $\mathcal{P}_{\pi}^i$  is the set of all players precede player  $i$  in permutation  $\pi$ , and  $[v(\mathcal{P}_{\pi}^i \cup \{i\}) - v(\mathcal{P}_{\pi}^i)]$  is marginal contribution of player  $i$  in permutation  $\pi$  [Shapley, 1953].

The aforementioned definition of contribution satisfies all the constraints mentioned above. Moreover, this is the only solution for evaluating contributions that has all these properties [Shapley, 1953].

### 2.2 Lagrange Coded Computing (LCC)

We now describe Lagrange coded computing (LCC) [18], a distributed coded computing method. Consider a system consisting of a master and  $M$  processing nodes. Suppose that  $f(x)$  is a polynomial function of degree  $d$ , and the master is interested in computing  $f(x)$  at  $K$  different private data points  $A_1, A_2, \dots, A_K$  (they may be either scalar, vector, or

matrix). Let  $R_1, R_2, \dots, R_T$  be  $T$  random data points of the same size as  $A_i$ 's, used to preserve the privacy of the  $A_i$ 's. To do that the master generates  $K+T$  distinct and random numbers  $\alpha_1, \alpha_2, \dots, \alpha_{K+T}$ . Then Polynomial  $P(x)$  is produced by Lagrange interpolation in a way that  $P(\alpha_i) = A_i, \forall i \in \{1, 2, \dots, K\}$  and  $P(\alpha_{K+i}) = R_i, \forall i \in \{1, 2, \dots, T\}$ . By using Lagrange interpolation, one can see that  $P(x)$  is as follows:

$$P(x) = \sum_{i=1}^K A_i \cdot \prod_{j=1, j \neq i}^{K+T} \frac{x - \alpha_j}{\alpha_i - \alpha_j} + \sum_{i=1}^T R_i \cdot \prod_{j=1, j \neq i+K}^{K+T} \frac{x - \alpha_j}{\alpha_{i+K} - \alpha_j} \quad (2)$$

Then, The master sends  $P(\beta_1), P(\beta_2), \dots, P(\beta_M)$  to the processing nodes  $1, 2, \dots, M$ , respectively, where  $\beta_i$ 's are some random elements of the field, distinct from the  $\alpha_i$ 's. Then, each processing node  $i$  computes  $f(P(\beta_i))$  and sends it back to the master. Note that the polynomial  $f(P(x))$  is of degree  $d(K+T-1)$ . Therefore, the master can recover the polynomial  $f(P(x))$  if it has access to  $d(K+T-1)+1$  points on it. Consequently, it can compute  $f(A_i)$ , which is equal to  $f(P(\alpha_i)), \forall i \in \{1, 2, \dots, K\}$ .

As shown in [18], LCC is  $T$ -private, meaning that even if  $T$  colluding nodes attempt to extract information about  $A_i$ , LCC ensures that no information is leaked. Additionally, the scheme is resilient to  $S$  straggler nodes that may fail to return computed data, as long as the master has access to  $S$  extra points of the polynomial—equivalent to having  $S$  additional processing nodes. Furthermore, if there are  $A$  adversarial nodes that may send incorrect responses, using  $2A$  additional processing nodes ensures both security and correctness through the Reed-Solomon error correction method. Thus, when the number of processing nodes satisfies  $M \geq d(K+T-1) + S + 2A + 1$ , the computation can be performed securely and privately.

## 3 Problem Setting

Consider a federated learning system comprising a master,  $N$  clients numbered 1 to  $N$ , and  $M$  processing nodes. Each client  $i$  has access to a labeled dataset  $D_i = (\mathcal{X}_i, \mathcal{Y}_i) = \{(x_{i,1}, y_{i,1}), \dots, (x_{i,n_i}, y_{i,n_i})\}$  of size  $n_i$ , where  $x_{i,j}$  is a feature vector and  $y_{i,j}$  is the corresponding label, for all  $i \in \{1, 2, \dots, N\}$ . Additionally, the master has a private evaluation dataset  $D = (\mathcal{X}, \mathcal{Y}) = \{(x_1, y_1), \dots, (x_n, y_n)\}$  of size  $n$ , used to assess the contributions of local models. The goal is to train a global model on the clients' datasets while also computing their individual contributions. The model is assumed to be a polynomial function. To determine the contributions, the master distributes the evaluation dataset  $D$  among the  $M$  processing nodes using secret sharing and calculates the Shapley value of lo-

cal models over them throughout  $T$  training rounds. The proposed scheme has the following steps:

- (1) **Sharing:** In this phase, the master shares dataset  $D$  among the  $M$  processing nodes before the training rounds. The key consideration in sharing datasets is to prevent clients from obtaining any information about the private dataset  $D$ . Otherwise, they could manipulate their model updates to unfairly increase their perceived contribution in training the final model.
- (2) **Local Training:** At training round  $t \in \{1, \dots, T\}$ , using the final model from the previous round, denoted as  $M^{(t-1)}$ , each client trains a local model by minimizing a specific loss function on its dataset. Clients then send their model updates to the master and processing nodes.
- (3) **Contributions Computing:** Following local training, processing nodes compute a contribution function over the coded dataset and transmit this information back to the master.
- (4) **Finding Shapley Value:** Finally, the master extracts each client's contribution from the coded contributions received from the processing nodes.
- (5) **Updating Model:** The master aggregates local models to compute a new global model  $M^{(t)}$  and broadcasts this updated model to clients for the subsequent iteration.

## 4 General Scheme

In this section, the proposed scheme is detailed. First, assume that we use polynomial models and there are  $T$  training rounds where at each round  $t \in \{1, \dots, T\}$ , the master sends the final model of the previous round,  $M^{(t-1)}$ , to each client. Each client  $i$  then trains a local model  $M_i^{(t)}$  on its local dataset  $D_i$  and subsequently sends the gradient update,  $\Delta_i^t$ , to the master, which is equal to:

$$\Delta_i^{(t)} = M_i^{(t)} - M^{(t-1)}$$

Then, the master updates the global model using the collected gradient vectors from the clients. To do this, the master computes a weighted average of the collected models, based on the size datasets as:

$$M^{(t+1)} = M^{(t)} + \sum_{i=1}^N \frac{n_i}{|D_N|} \Delta_i^{(t+1)} \quad (3)$$

where  $n_i$  denotes size of  $D_i$  and  $|D_N| = \sum_i n_i$ . Now, let  $S$  be a subset of the locally trained models. We define  $M_S$  as:

$$M_S = M^{(t)} + \sum_{i \in S} \frac{n_i}{\sum_{j \in S} n_j} \Delta_i^{(t+1)}$$

i.e.,  $M_S$  is the global model in the case that the master receives local models just from the subset  $S$ . According to Equation 1, the contribution of each client is measured by the Shapley value as follows:

$$\phi_i = \sum_{S \in \{1,2,\dots,N\} \setminus \{i\}} \frac{v(S \cup \{i\}) - v(S)}{\binom{N-1}{|S|}} \quad (4)$$

where  $v(S)$  is the payoff function of model  $M_S$ . For a single test data  $(x, y)$ ,  $M_S(x)$  denotes the corresponding predicted label of  $x$  using model  $M_S$ . Let  $g(y, \hat{y})$  be a goodness function, indicating the proximity of  $y$  and  $\hat{y}$ , e.g.,  $-(y - \hat{y})^2$ . Then, for a dataset  $D$ , the payoff function  $v(S)$  is defined as:

$$v(S) = \sum_{(x,y) \in D} g(y, M_S(x)) \quad (5)$$

By (4) and (5), computing clients' contribution has two challenges. First, the Shapley value requires computing the marginal contribution for each subset of clients, which is computationally expensive. Second, computing the payoff function for each subset of clients requires summing the function  $g$  over the entire evaluation dataset  $D$ . Therefore, in practice, computing marginal contributions is almost impossible.

To address the first challenge, methods based on Monte Carlo estimation have been proposed. Consider a permutation  $\pi : p_1, p_2, \dots, p_N$ . Define  $S_\pi^i$  as the set of clients that precede client  $p_i$  in the permutation. Then, Equation 4 can be written as:

$$\phi_i = \mathbb{E}_{\pi \in \Pi} [v(S_\pi^i \cup \{i\}) - v(S_\pi^i)]$$

In these methods, instead of computing the marginal contribution for each client across all permutations, we randomly sample permutations. Via the Monte Carlo estimation method, the master samples  $K$  permutations  $\pi_1, \pi_2, \dots, \pi_K$  from  $\Pi(N)$  randomly and computes an estimate of the contributions as follows:

$$\phi_i \approx \frac{1}{K} \sum_{k=1}^K [v(S_{\pi_k}^i \cup \{i\}) - v(S_{\pi_k}^i)] \quad (6)$$

To be able to compute payoff function  $v(S)$  over dataset  $D$ , the master computes models  $M_{S_{\pi_k}^i}$  for each sampled permutation  $\pi_k, k \in \{1, 2, \dots, K\}$  and each  $i \in \{1, 2, \dots, N\}$ . Then, it sends models  $M_{S_{\pi_k}^1}, \dots, M_{S_{\pi_k}^N}$  to the processing nodes.

To deal with the second challenge, the master secret-shares the evaluation dataset  $D$  among the  $M$  processing nodes. Note that this dataset must be kept private, as the clients may receive payment due to their participation in training the final model. If a client learns any information about the private dataset  $D$ , it might train its local model based on

this information to increase its contribution and receive more rewards. To maintain privacy, we employ LCC to share the dataset  $D$ . Assume that the size of dataset  $D$ ,  $n$ , is the product of two natural numbers  $a$  and  $b$ , i.e.  $n = ab$ . The master partitions  $D$  into  $b$  datasets,  $D'_1, D'_2, \dots, D'_b$  each of size  $a$  as follows:

$$D'_i = \{(x_{(i-1)a+1}, y_{(i-1)a+1}), \dots, (x_{ia}, y_{ia})\}$$

Then, it constructs polynomials  $P_1(x), \dots, P_b(x)$  according to (2), such that:

$$P_1(\alpha_1) = (x_1, y_1), \dots, P_1(\alpha_a) = (x_a, y_a)$$

$$P_2(\alpha_1) = (x_{a+1}, y_{a+1}), \dots, P_2(\alpha_a) = (x_{2a}, y_{2a})$$

⋮

$$P_b(\alpha_1) = (x_{(b-1)a+1}, y_{(b-1)a+1}), \dots, P_b(\alpha_a) = (x_{ba}, y_{ba})$$

To preserve the privacy against  $u$  colluding processing nodes, it adds some random points to each polynomial as follows:

$$P_l(\alpha_{a+1}) = R_{l,1}, \dots, P_l(\alpha_{a+u}) = R_{l,u}, \forall l \in \{1, 2, \dots, b\}$$

where  $R_{l,t}$ s are random data points with the same size as  $(x, y)$  in  $D$ . According to [18], one can see that  $P_l(x)$  is of degree  $a + u - 1$ . For each  $l \in \{1, 2, \dots, b\}$  the master sends  $P_l(\beta_j) = (\tilde{x}_{l,j}, \tilde{y}_{l,j})$  to processing node  $j$ , where  $\beta_j$  is a random number assigned to node  $j$ .

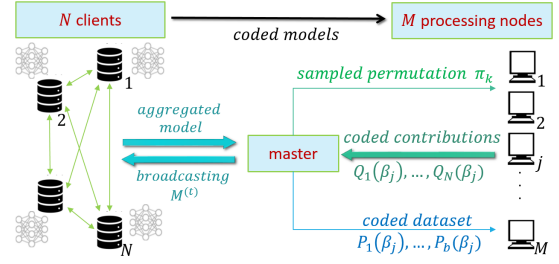
To compute Shapley value, each processing node  $j$  computes  $M_{S_{\pi_k}^i}(\tilde{x}_{l,j})$  for all  $i, l$ , and sampled permutation  $\pi_k$ . In other words, for permutation  $\pi_k$  and client  $i \in \{1, 2, \dots, N\}$ , node  $j$  computes  $b$  values as follows:

$$\hat{y}_1^j = M_{S_{\pi_k}^i}(\tilde{x}_{1,j}), \dots, \hat{y}_b^j = M_{S_{\pi_k}^i}(\tilde{x}_{b,j})$$

One can see that if functions  $M_{S_{\pi_k}^i}(x)$  were polynomial functions of degree  $r$ , values of  $\hat{y}_l^j$  would lie on a polynomial of degree  $r(a + u - 1)$ . The processing nodes then compute the goodness function  $g(\tilde{y}_{l,j}, \hat{y}_l^j)$ . Let us assume that  $g$  is a function of degree  $s$ . As a result, for each permutation  $\pi_k$ , each client  $i$ , and each polynomial  $P_l(x)$ , the values  $g(\tilde{y}_{l,j}, M_{S_{\pi_k}^i}(\tilde{x}_{l,j}))$  will lie on a polynomial of degree  $rs(a + u - 1)$  like  $Q_{i,l}(\beta)$ . By this point, each processing node  $j$  has access to  $Q_{i,1}(\beta_j), \dots, Q_{i,b}(\beta_j)$ . According to the properties of LCC, we know that for each  $j \in \{1, 2, \dots, a\}$ :

$$\begin{aligned} Q_{i,1}(\alpha_j) &= g(y_j, M_{S_{\pi}^i}(x_j)) \\ Q_{i,2}(\alpha_j) &= g(y_{a+j}, M_{S_{\pi}^i}(x_{a+j})) \\ &\vdots \\ Q_{i,b}(\alpha_j) &= g(y_{(b-1)a+j}, M_{S_{\pi}^i}(x_{(b-1)a+j})) \end{aligned} \quad (7)$$

According to (5),  $v(S_{\pi}^i) = \sum_{(x,y) \in D} g(y, M_{S_{\pi}^i}(x))$ . Thus, by (7), it can be written as:



**Figure 1.** Proposed scheme (The master shares the evaluation dataset with each processing node  $j$  by sending  $P_1(\beta_j), \dots, P_b(\beta_j)$ . At each round, each client trains a local model and sends it to the processing nodes. Then, processing node  $j$  computes the coded contribution of each client for permutation  $\pi_k$  and sends them back to the master by  $Q_1(\beta_j), \dots, Q_N(\beta_j)$ . Finally, the master interpolates these polynomials and computes the contributions)

$$v(S_{\pi}^i) = \sum_{j=1}^a \sum_{l=1}^b Q_{i,l}(\alpha_j) = \sum_{j=1}^a Q_i(\alpha_j) \quad (8)$$

where  $Q_i(x) = \sum_{l=1}^b Q_{i,l}(x), \forall i \in \{1, 2, \dots, N\}$ . Then, each node  $j$  computes  $Q_i(\beta_j) = Q_{i,1}(\beta_j) + \dots + Q_{i,b}(\beta_j)$  and sends it to the master. Due to the linearity of LCC, one can see that  $Q_i(\alpha_j) = Q_{i,1}(\alpha_j) + \dots + Q_{i,b}(\alpha_j)$  for  $i \in \{1, 2, \dots, N\}$ , and  $j \in \{1, 2, \dots, a\}$ .

Finally, for each permutation  $\pi_k$  the master interpolates the polynomial  $Q_i(x)$  for  $i \in \{1, \dots, n\}$ , and computes the permutation payoff functions  $v(S_{\pi_k}^i) = Q_i(\alpha_1) + \dots + Q_i(\alpha_a)$  according to (8). Then, it computes each node's marginal contribution for the given permutation as follows:

$$\phi_i^{(k)} = [v(S_{\pi_k}^i \cup \{i\}) - v(S_{\pi_k}^i)]$$

After computing these marginal contributions for all  $\pi_1, \pi_2, \dots, \pi_K$ , the master will announce  $\phi_i \approx \frac{\sum_k \phi_i^{(k)}}{K}$  as the contribution of client  $i$  according to (6). Since we train the global model in  $T$  rounds, the final contribution of each client would be the sum of its contributions from each round.

The general scheme of the proposed algorithm is depicted in Figure 1, and its pseudocode is available in Pseudocode 1.

Now, we want to determine the number of processing nodes required for the scheme. To interpolate the polynomial  $Q_i(x)$  of degree  $rs(a + u - 1)$ , the master must have its value on at least  $rs(a + u - 1) + 1$  points. Therefore, the total number of processing nodes must satisfy  $M \geq rs(a + u - 1) + 1$ . For instance, if the FL model is linear and the goodness function is the squared error on the evaluation dataset, then  $r = 1$  and  $s = 2$ . In this scenario, we need at least  $2a + 2u - 1$  processing nodes. Moreover, according to LCC properties by adding  $S + 2A$  additional nodes, we can

**Pseudocode 1** Distributed Contribution-Based FL**Parameters:**  $a, b$ 


---

```

1: Initialization :
2: the master has an initial model  $M^{(0)}$ .
3: the master shares  $D$  with processing nodes by LCC.
4: Training Rounds:
5: for  $t = 1$  to  $T$  do
6:   the master sends  $M^{(t-1)}$  to the clients.
7:   each client  $i$  computes local model  $M_i^{(t)}$ .
8:   each client  $i$  sends  $\Delta_i^{(t)} = M_i^{(t)} - M^{(t-1)}$  to the processing nodes.
9:   the master generates  $K$  random permutations  $\pi_1, \dots, \pi_K$ .
10:  for  $k = 1$  to  $K$  do
11:    the master sends  $\pi_k$  to the processing nodes.
12:     $v(\emptyset) = \sum_{(x,y) \in D} g(y, M^{(t-1)}(x))$ .
13:    for  $i = 1$  to  $N$  do
14:      processing node  $j$  computes  $Q_i(\beta_j)$ .
15:      node  $j$  sends  $Q_i(\beta_j)$  to the master.
16:      the master interpolates  $Q_i(x)$ .
17:       $v(S_{\pi_k}^i) = Q_i(\alpha_1) + \dots + Q_i(\alpha_a)$ .
18:       $\phi_i^{(k)} = [v(S_{\pi_k}^i \cup \{i\}) - v(S_{\pi_k}^i)]$ .
19:    end for
20:  end for
21:  client  $i$  contribution:  $\phi_i(t) \approx \sum_k \frac{\phi_i^{(k)}}{K}$ .
22:  the master aggregates  $\Delta_i^{(t)}$ 's and computes  $M^{(t)}$ .
23: end for

```

---

make the system resilient to  $S$  stragglers and secure against  $A$  adversaries, as mentioned in Section 2.

## 5 Evaluation of the Proposed Scheme

In this section, we compare the proposed method with the previous ones in terms of security, privacy, processing time, and computational complexity.

By calculating the computational complexity, as explained in the appendix, we find that our scheme has less computational complexity than the conventional contribution-based FL, where the computations are not distributed. Moreover, we can optimize the computation by adjusting parameters  $a$  and  $b$ . In fact,  $a$  and  $b$  are design parameters for balancing the computations between the master and processing nodes. For example, in cases where the model is too complex, the computational complexity of processing nodes dominates that of the master. In such cases, we set  $b = 1$  to reduce the algorithm's complexity, which is equivalent to coding the entire dataset  $D$  in a single polynomial. Additionally, in cases where the number of processing nodes is limited, we can adjust

$a$  to satisfy  $M \geq rs(a + u - 1) + 1$ , which is equivalent to increasing  $b$  and the master's computational complexity. In another simple scenario, we can set  $a = b = \sqrt{n}$ . In this case, the computational complexity of the master for decoding the contributions is approximately  $\mathcal{O}(KNrs\sqrt{n}(\log rs\sqrt{n})^2)$  and the computational complexity of each processing node is  $\mathcal{O}(KNw\sqrt{n})$  at each round (refer to the appendix for more details). Therefore, the total complexity is around  $\mathcal{O}(TKNrs\sqrt{n}(\log rs\sqrt{n})^2)$ , which is much more efficient than the conventional SV computation ( $\mathcal{O}(KNwn)$ ) in terms of  $n$ . However, the complexity is linear in terms of  $N$ , which could still be high in large systems with numerous clients. Hence, reducing the complexity to a sublinear level in terms of  $N$  is a subject for future research.

The privacy-preserving proof of this work for the evaluation dataset is attributed to the privacy of LCC. According to [18], this sharing method is information-theoretically private against  $u$  colluding node. In contribution-based federated learning, the privacy of local models is often not preserved since the contributions are computed based on them. The distributed structure of our model allows clients to encode their local models before sending them to processing nodes. For example, consider one of the model's coefficients as  $g$ . A client can encode it with a polynomial of degree  $u$  like  $G(x)$  using LCC, such that for  $j \in \{1, 2, \dots, a\}$ ,  $G(\alpha_j) = g$ . Then it sends  $G(\beta_j)$  for processing node  $j$ . After that, the processing nodes continue the computation using  $G(\beta_j)$  instead of coefficient  $g$  in the main algorithm. Thus, we are also able to preserve the local models' privacy.

As mentioned earlier, our system is private against colluding nodes, resilient against stragglers, and secure against adversaries who may send wrong answers. However, there may still be byzantine clients who send incorrect model updates, causing deviations in the global model coefficients. This impact can be mitigated by omitting the gradient update vectors that significantly differ from others using algorithms like k-rum [19]. However, in contribution-based federated learning systems, detecting byzantine clients is straightforward, as their contributions are very small or even negative. Thus, after computing contributions, the master can easily identify and exclude byzantine clients during aggregation, making the proposed scheme resistant to byzantine clients, as noted in [20].

**Remark 1.** This method was designed for polynomial FL models, such as linear regression with polynomial goodness functions. However, it can be applied to more general cases using polynomial approximation. For instance, if the goodness function is defined

by the absolute error between evaluation dataset labels and predicted labels by a trained model, i.e.  $g(y, \hat{y}) = -|y - \hat{y}|$ , assuming that the model output is uniformly in  $[-1, 1]$ , a 6th-degree approximation function  $-3.076x^2 + 4.512x^4 - 2.514x^6$  can be obtained through squared error minimization. There are some approaches to approximating a learning model with a polynomial. For example, [21] employed polynomial approximation of logistic regression by least square estimation of the sigmoid function to train the model in a distributed manner. However, polynomial approximation of complex models results in high-degree polynomials, which subsequently slow down the decoding phase and increase the number of processing nodes. Hence, designing an efficient SV computing method for non-polynomial models is a task for future research.

**Remark 2.** In this setting, we separate processing nodes from the clients. In cases where the clients have acceptable processing power, there is no need for additional processing nodes, and the computations can be performed by the clients. Since the data sent to the nodes is encoded using LCC, there is no concern about information leakage, and the clients cannot obtain any information about the evaluation dataset from the encoded data.

**Remark 3.** According to references [14, 17], SV has various applications in different areas of machine learning, such as feature selection and attack detection. Our proposed method can be implemented in these applications to enhance computational efficiency. Additionally, there are different architectures of Federated Learning, noted in [22]. For instance, several recent studies have leaned toward decentralized architectures that eliminate the need for a central server. Therefore, developing an efficient contribution assessment based on SV in such architectures could be an intriguing direction for future research.

## 6 Conclusion

This study introduces a contribution-based federated learning scheme that efficiently computes a learning model and clients' contributions compared to existing methods. Computing clients' contributions is essential for ensuring their reliability, as they are incentivized to use real datasets. To expedite the contribution computation, we privately distribute the master's dataset using LCC. We evaluated the proposed scheme in terms of computational complexity, processing time, privacy, and handling byzantine clients, in comparison to existing schemes. Finally, we highlight the importance of reducing complexity in terms of the number

of clients, SV computation for non-polynomial models, and evaluating SV in decentralized architectures as potential future research directions.

## References

- [1] Konstantin Sozinov, Vladimir Vlassov, and Sarunas Girdzijauskas. Human activity recognition using federated learning, 2018.
- [2] Joel Stremmel and Arjun Singh. Pretraining federated text models for next word prediction. 2020.
- [3] Fan Bai, Jiayang Wu, Pengcheng Shen, Shaoxin Li, and Shuigeng Zhou. Federated face recognition. volume abs/2105.02501, 2021.
- [4] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *CoRR*, abs/1908.07873, 2019.
- [5] Ashish Rauniyar, Desta Haileselassie Hagos, Debesh Jha, Jan Erik Håkegård, Ulas Bagci, Danda B Rawat, and Vladimir Vlassov. Federated learning for medical applications: A taxonomy, current trends, challenges, and future research directions. *IEEE Internet of Things Journal*, 2023.
- [6] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys*, 23(3):1622–1658, 2021.
- [7] Yao Chen, Yijie Gui, Hong Lin, Wensheng Gan, and Yongdong Wu. Federated learning attacks and defenses: A survey. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 4256–4265, 2022.
- [8] Jinhyun So, Ramy E. Ali, Basak Guler, Jiantao Jiao, and Salman Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning, 2021.
- [9] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis, 2020.
- [10] Ahmed El Ouadrhiri and Ahmed Abdelhadi. Differential privacy for deep and federated learning: A survey. *IEEE Access*, 10:22359–22380, 2022.
- [11] Tayyeb Jahani-Nezhad, Mohammad Ali Maddah-Ali, and Giuseppe Caire. Byzantine-resistant secure aggregation for federated learning based on coded computing and vector commitment. *arXiv: 2302.09913*, 2023.
- [12] Yuxin Shi, Han Yu, and Cyril Leung. Towards

fairness-aware federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–17, 2023.

- [13] Guan Wang, Charlie Xiaoqian Dang, and Ziyue Zhou. Measure contribution of participants in federated learning, 2019.
- [14] Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Olivér Kiss, Sebastian Nilsson, and Rik Sarkar. The shapley value in machine learning. *arXiv*, 2022.
- [15] Zelei Liu, Yuanyuan Chen, Han Yu, Yang Liu, and Lizhen Cui. Gtg-shapley: Efficient and accurate participant contribution evaluation in federated learning. *arXiv*, 2021.
- [16] Ziwen Cheng, Yi Liu, Chao Wu, Yongqi Pan, Liushun Zhao, and Cheng Zhu. *PoShapley-BCFL: A Fair and Robust Decentralized Federated Learning Based on Blockchain and the Proof of Shapley-Value*, pages 531–549. 11 2023.
- [17] Liguang Dong, Zhenmou Liu, Kejia Zhang, Abdulsalam Yassine, and M. Shamim Hossain. Affordable federated edge learning framework via efficient shapley value estimation. *Future Generation Computer Systems*, 147:339–349, 2023.
- [18] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A. Avestimehr. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1215–1225. PMLR, 16–18 Apr 2019.
- [19] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Byzantine-tolerant machine learning. *arXiv*, 2017.
- [20] Vincent Labatut Khaoula Otmani, Rachid Elazouzi. Fedsv: Byzantine-robust federated learning via shapley value. In *IEEE International Conference on Communications*, Jun 2024.
- [21] Jinhyun So, Başak Güler, and A. Salman Avestimehr. Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):441–451, 2021.
- [22] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. Federated learning systems: Architecture alternatives. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 385–394, 2020.
- [23] Kiran S Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.



**Mohammad Amin Sarzaeem** obtained his B.Sc. degree in Electrical Engineering (Communication Systems) with a minor in Mathematics from Sharif University of Technology, Tehran, Iran, in 2023. He is currently pursuing an M.Sc. degree in Communication Systems at Sharif University of Technology, Tehran, Iran. Since 2022, Mohammad has been affiliated with the Information Systems and Security Lab (ISSL) at Sharif University of Technology. His research interests include Information Theory, Machine Learning, and Distributed Computing.



**Seyed Reza Hoseini Najarkolaei** is currently a Ph.D. student in Communication Systems at Sharif University of Technology. He received his M.Sc. and B.Sc. degrees in Communication Systems from the same university in 2020 and 2017, respectively.

His research interests include network coding, cryptography, and data privacy. He has published several papers in peer-reviewed journals and conferences in the fields of coded computing and information theory.



**Mohammad Reza Aref** received the B.Sc. degree in 1975 from the University of Tehran, Iran, and the M.Sc. and Ph.D. degrees in 1976 and 1980, respectively, from Stanford University, Stanford, CA, USA, all in electrical engineering. He returned to

Iran in 1980 and was actively engaged in academic affairs. He was a Faculty member of Isfahan University of Technology from 1982 to 1995. He has been a Professor of electrical engineering at the Sharif University of Technology, Tehran, since 1995. His current research interests include areas of Communication Theory, Information Theory, and Cryptography.

## A Computational Complexity of the Algorithm

In [23], it is shown that the computational complexity of interpolating a polynomial of degree  $q$  is  $\mathcal{O}(q \log q^2 \log \log q)$ . Also, evaluating the polynomial at  $q$  points requires the same computation complexity as interpolation. Therefore, evaluating a polynomial at  $M$  points has a complexity of  $\mathcal{O}(M \log q^2 \log \log q)$ . To compute the total complexity of the scheme, we have to calculate the complexity of each part separately:

- In the sharing phase, the master interpolates polynomials  $P_1(x), \dots, P_b(x)$  of de-

gree  $d = a + u - 1$  with computation complexity of  $\mathcal{O}(bd \log d^2 \log \log d)$ . Then for  $j \in \{1, 2, \dots, M\}$  it calculates the value of them on  $\beta_j$  and sends them to processing node  $j$  with a computational complexity of  $\mathcal{O}(\frac{M}{d}bd \log d^2 \log \log d) \simeq \mathcal{O}(rsb \log d^2 \log \log d)$ .

- In the contribution assessment phase, in each round, for each permutation  $k$  and each client  $i$ , each processing node  $j$  computes model  $M_{S_{\pi_k}^i}$  output on their  $b$  data points and calculates the values of  $Q_{i,1}(\beta_j), Q_{i,2}(\beta_j), \dots, Q_{i,b}(\beta_j)$ . As the model complexity varies in different applications, we denote it as  $\mathcal{O}(w)$ . So, processing nodes' computations have a complexity of  $\mathcal{O}(wb)$ .
- In the contribution assessment phase, in each round, for each permutation and each client, the master interpolates polynomial  $Q_i(x)$  of degree  $rsd$  with a computational complexity of  $\mathcal{O}(rsd \log rsd^2 \log \log rsd)$ .

This complexity analysis is summarized in [Table A.1](#).

The sharing phase is implemented once at the beginning to share the evaluation dataset, whereas the last two items must be implemented several times, in each round, for each permutation and each client ( $TKN$  times). So, the last two phases are computationally dominant with complexity of

**Table A.1.** computational complexity: since  $d \simeq a$  and  $ab = n$  is constant, the computations between master and processing nodes can be balanced to speed up the process.

	computation
master (sharing phase)	$\mathcal{O}(rsb \log d^2 \log \log d)$
master (contribution assessment phase)	$\mathcal{O}(TKNrsd \log rsd^2 \log \log rsd)$
processing nodes	$\mathcal{O}(TKNwb)$

$\mathcal{O}(TKN \max(wb, rsd \log rsd^2 \log \log rsd))$ . Since  $ab = n$  is constant, we can choose  $a$  and  $b$  to minimize the complexity. If we relax the problem to minimizing  $\mathcal{O}(TKN \max(wb, (rsd)^2))$ , under the assumption of  $d \simeq a$ , the optimal solution is  $\mathcal{O}(TKN(rswn)^{\frac{2}{3}})$  by setting  $a = (wn/rs^2)^{\frac{1}{3}}$  and  $b = ((nrs)^2/w)^{\frac{1}{3}}$ .

In conventional Shapley value-based contribution assessment methods, in each round, for each permutation and each client, the master computes the model  $M_{S_{\pi_k}^i}$  output on  $n$  data points and computes  $v(S_{\pi_k}^i)$  through (5), which has a complexity of  $\mathcal{O}(TKNwn)$ . For large  $n$ , we conclude that the proposed scheme is significantly faster than the conventional Shapley value contribution-based FL. Additionally, it is possible to further accelerate our model at the cost of additional processing nodes. This can be achieved by partitioning  $D$  into multiple groups, with each group's computations handled by a distinct set of processing nodes.