Secure Coalition Formation for Federated Machine Learning

Subhasis Thakur(✉) and John Breslin

University of Galway, Galway, Ireland
{subhasis.thakur,john.breslin}@universityofgalway.ie

Abstract. Federated machine learning allows multiple collaborative parties to build a machine learning model in a distributed fashion where the data is distributed among the parties. There can be few restrictions on the participation of such parties in a federate machine learning process. For example, parties may have spatial restrictions such that only a limited number of parties from one locality can participate in one federated machine learning process. Only a fixed number of parties may participate in one federated machine-learning process. To include such restrictions on federated machine learning we need to consider optimally of such group formation. This optimization condition will ensure that each group's aggregated DNN model reaches sufficient accuracy. In this paper, we investigate such constraints on the federated machine learning process. Specifically, we explore party-affiliation game-based partition over the parties for a federated machine learning model. Partition function games find optimal and stable partitions (where no party is better off by changing groups to improve the accuracy of the aggregated DNN model) among the parties and one federated machine learning model is formed in each partition. Further, we investigate methods for secure and privacy-preserving federated machine learning processes where parties are partitioned into a set of groups and the machine learning model of one party is not shared with other parties in different groups. Our main contributions are (a) we proposed a federated machine learning protocol for a group of participants where the number of parties in each group may not exceed a predefined threshold, (b)we proposed a federated machine learning protocol that forms a stable partition among the parties such that no party can leave a group and join another group to improve the accuracy of aggregated DNN models, (c) Our proposed federated machine learning protocol is secure as it prevents a party from reporting the wrong DNN model to the aggregator, (d)our proposed federated machine learning protocol is secure as ensures that the aggregator correctly generates the coalition formation protocol, (e)our proposed federated machine learning protocol is secure and privacy-preserving as it ensures that valid DNN model parameters are exchanged among the parties.

Keywords: Federated machine learning · Zero knowledge proofs · zkSNARK

1 Introduction

Federated machine learning allows multiple collaborative parties to build a machine learning model in a distributed fashion where the data is distributed among the parties. The advantages of federated machine learning are (a) data can be kept and owned

A. Fred et al. (Eds.): DeLTA 2024, CCIS 2171, pp. 238–258, 2024.
https://doi.org/10.1007/978-3-031-66694-0_15

locally by the parties and they can have the privacy of their data, and (b) computation of machine learning models from the data can be distributed among the parties and hence this approach of machine learning model formation have less computation load than centralised machine learning.

The federated machine learning process is cooperative in the sense that parties cooperate and share the machine learning models to aggregate them into a better machine learning model often using a trusted aggregator. Usually, in such a federated machine learning process parties have no restriction on contributing towards an aggregated machine learning model.

However, there can be few restrictions on the participation of such parties in a process. For example, parties may have spatial restrictions such that only a limited number of parties from one locality can participate in one federated machine learning process. Only a fixed number of parties may participate in one federated machine-learning process. For example, assume that a vehicle generates a Deep Neural Network (DNN) model for traffic congestion in a specific locality as the vehicle is used in a specific geographic location. Given a set of vehicles with DNN models for traffic congestion at various localities, suppose that subsets of vehicles want to combine their DNN models to generate congestion over multiple geographic locations then there is a restriction on the membership of such a subset. For example, we may want to add one vehicle for each geographic location. Hence various groups of vehicles can form groups to aggregate their DNN models. However, to include such restrictions on federated machine learning we need to consider optimally of such group formation. This optimization condition will ensure that each group's aggregated DNN model reaches sufficient accuracy. In this paper, we investigate such constraints on the federated machine learning process. Specifically, we explore party-affiliation game-based partition over the parties for a federated machine learning model. Partition function games find optimal and stable partitions (where no party is better off by changing groups to improve the accuracy of the aggregated DNN model) among the parties and one federated machine learning model is formed in each partition.

Further, we investigate methods for secure and privacy-preserving federated machine learning processes where parties are partitioned into a set of groups and the machine learning model of one party is not shared with other parties in different groups. Our main contributions are as follows:

1. We proposed a federated machine learning protocol for a group of participants where the number of parties in each group may not exceed a predefined threshold.
2. We proposed a federated machine learning protocol that forms a stable partition among the parties such that no party can leave a group and join another group to improve the accuracy of aggregated DNN models.
3. Our proposed federated machine learning protocol is secure as it prevents a party from reporting the wrong DNN model to the aggregator.
4. Our proposed federated machine learning protocol is secure as ensures that the aggregator correctly generates the coalition formation protocol.
5. Our proposed federated machine learning protocol is secure and privacy-preserving as it ensures that valid DNN model parameters are exchanged among the parties.

6. We present mathematical proofs of security and privacy-preserving properties of the proposed federated machine learning model.
7. We present an experimental evaluation of the security and privacy-preserving properties of the proposed federated machine learning model.

The paper is organized as follows: in Sect. 2 define the federated machine learning protocol development problems, in Sect. 3, we discuss related literature, in Sect. 4, we present the federated machine learning protocols, in Sect. 5, we analyze equilibrium and privacy-preserving properties of these protocols, in Sect. 6, we present experimental evaluation and we conclude the paper in Sect. 7.

2 Problem Statement

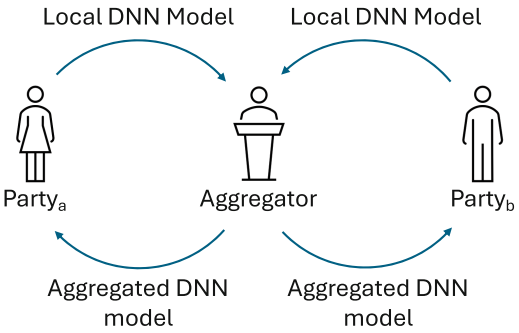

Fig. 1. Federated Machine Learning Procedure.

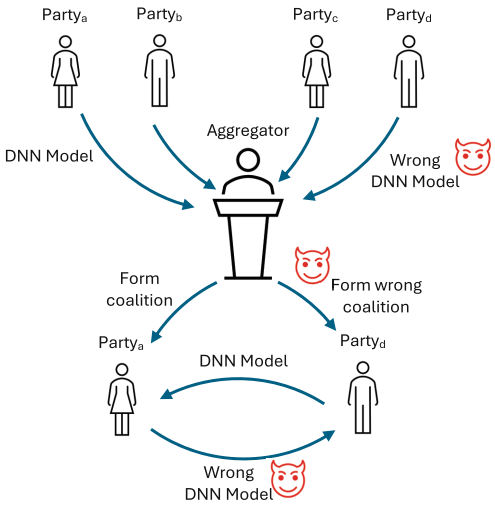

Fig. 2. Security problems with coalition formation for federated Machine Learning Procedure.

In a federated machine learning setting, we have several parties develop their own local DNN model and an aggregator combines these local DNN models. However, coalition

formation for federate machine learning requires the aggregator to determine a partition over the parties, i.e., a set of coalitions, and one aggregated coalition is formed in each coalition. There are several trust problems in combining such local DNN models:

Trust Problem 1. A party may not share the correct DNN function evaluation with the aggregator to evaluate the accuracy of its DNN model. We need to verify that the classification result of a DNN model is a result of the DNN model execution given specific input data.

Trust Problem 2. We intend to use a partition over the parties into groups where one DNN model is aggregated in each group. The parties need to trust to aggregator to correctly execute such a partition algorithm.

Trust Problem 3. We need to ensure if a party reports its DNN model to another party then the reported DNN parameter corresponds to the verified DNN model.

Trust Problem 4: In coalition-based federated machine learning, the parties are partitioned into groups or coalitions and one DNN model is generated by aggregating the local DNN model in each group. However, a party may not trust the partitioning algorithm as it may be better off by switching coalitions.

3 Related Literature

In [19] authors explored a new cooperation method for auction-based federated learning. In [17] proposed a federated machine learning method without a centralised aggregator. In [13] the authors have developed a method for model aggregation for federated learning by averaging model parameters. In [22] the authors improved average model aggregation by imposing a range over the model parameters. [2] explored homomorphic encryption for privacy-preserving model aggregation federated machine learning. In [21] the authors explored differential privacy for privacy-preserving aggregation of machine learning models. In [11] the authors have developed a hierarchical aggregation of machine learning models. In [9] the authors have investigated issues with average aggregation of machine learning models. In [6] proposed a communication optimised federated machine learning protocol. In [16] the authors explored adaptive optimisation for federated machine learning. In [20] the authors have developed federated machine learning for convolutional neural networks (CNNs) and LSTMs. [12] explored security issues with federated machine learning. In [7] the authors have developed a decentralised federate machine learning protocol. In [23] the authors have developed a federated machine learning protocol using a trusted execution environment. In [3] the authors have developed secure model aggregation while using heterogeneous quantization. In [18] the authors have developed a privacy-preserving model aggregation for federated machine learning.

In this paper, we used zero-knowledge proof for DNN verification. In [4] the authors have developed a zero-knowledge proof protocol that allows the prover to prove that it still knows a number. We use Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) which reduces the size of proof and complexity of proof verification considerably. We use the KGZ polynomial commitment scheme developed in [8]. Other constructions of zkSNARKs developed in [5,10,14] use quadratic arithmetic program-based construction of zkSNARK.

4 Coalition Formation for Federated Machine Learning

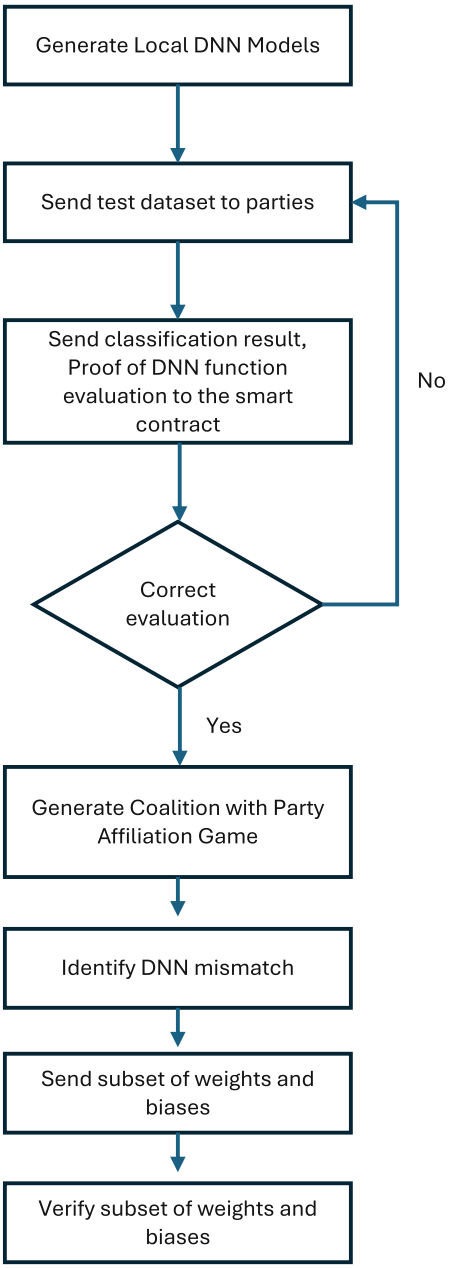

Fig. 3. Overview of the proposed federated machine learning procedure.

Briefly, our approach to secure coalition-based federated machine learning procedure is as follows:

1. Parties generate their local DNN models. (Described in Sect. 4.1)
2. There is an aggregator (a smart contract in the Ethereum Blockchain network) with a set of test data where the classification results of the test data are not known to the parties.
3. The aggregator sends the test data to all parties and each party returns the following to the aggregator:
   (a) Classification result for the test data.
   (b) A proof of evaluation for the test data that includes a commitment to the DNN model, and a commitment to the DNN function evaluation.
4. The aggregator verifies the proof of evaluation of DNN models. (Described in Sect. 4.2)
5. If such verification holds then it generates a partition over the parties according to party affiliation games. The aggregator informs the parties about the coalitions. (Described in Sect. 4.3)
6. In each coalition parties identify subsets of weights and biases which have sufficient mismatches using a multi-party computation protocol. (Described in Sect. 4.4)
7. The parties exchange the identified subsets of weights and biases and verify if such values are correct using oblivious transfer protocol. (Described in Sect. 4.5)

4.1 Local DNN Model Computation

A DNN model with k layers and n nodes in each layer is as follows (Table 1):

Table 1. DNN model.

Input	a_0
	$\dots$
	a_n
$Layer_1$	$y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1$
	$\dots$
	$y_n^1 = b_n^1 + \cup_{i=1}^{n} a_i w_{i,n}^1$
$Layer_2$	$y_1^2 = b_1^2 + \cup_{i=1}^{n} a_i w_{i,1}^2$
	$\dots$
	$y_n^2 = b_n^2 + \cup_{i=1}^{n} a_i w_{i,n}^2$
$\dots$	$\dots$
$Layer_k$	$y_1^k = b_1^k + \cup_{i=1}^{n} a_i w_{i,1}^k$
	$\dots$
	$y_n^k = b_n^k + \cup_{i=1}^{n} a_i w_{i,n}^k$

In the above DNN model, the j'th node at $layer_i$ computes the following function

$$y_j^i = b_j^i + \cup_{x=1}^{n} y_j^{i-1} w_{x,j}^i$$

where all parameters are real numbers. All functions y_j^i will be referred to as DNN functions. An activation function modifies the outcome of each DNN function to be used as input to the next layer. We assume that the activation function is Relu, i.e., if the outcome of a DNN function is positive then its value remains the same otherwise it becomes zero. Given an input $(a_0, \ldots, a_n)$, the activation function will modify the outcome of a DNN function as either 0 or > 0. We will build a binary representation from such an activation function. If the outcome from the activation function is 0 then it will represent binary 0 otherwise binary 1.

4.2 Verification of DNN Models

A party does not reveal to the aggregator the DNN function for input as it wants to preserve the privacy of its DNN model. Instead, it will engage in a zero-knowledge proof protocol to verify that the binary representation is valid for the valuation of its DNN functions. We will verify a binary DNN model using KGZ polynomial commitment scheme-based zkSNARK. We will refer to the party with a DNN model and DNN function value for an input. We will refer to the aggregator who wants to verify that a binary representation of a DNN function is valid without the knowledge of the DNN function value as the verifier. In the above DNN there are $n * k$ functions where each function has degree n if we represent each function as follows:

$$f(x) = y_1^1 = b_1^1 + \cup_{i=1}^{n} a_i w_{i,1}^1 \tag{1}$$

$$= b_1^1 + a_1 x^1 + a_2 x^2 + \cdots + a_n x^n \tag{2}$$

where x represent the set of inputs $a_1, \ldots, a_n$. A proof of execution for the DNN will consist of the following:

1. A **commitment to the function** $f(x)$ that can hide the weights and biases to the verifier.
2. **Value of the function evaluation**, i.e., value of $y_1^1 = f(x)$ where x represents the set of inputs. For functions representing nodes from layer 2 to layer k, inputs will be the value of function evaluation such as y_1^1.
3. A **proof of evaluation** that can prove that y_1^1 indeed the result of executing $f(x)$ with the input $a_1, \ldots, a_n$.

A basic protocol for executing a proof of execution is as follows:

1. There are two parties, the prover(who has the DNN model) and the verifier(who has the input to the DNN model). The verifier wants to verify that given the output of the DNN model for input data, the output is generated by executing the DNN functions(i.e., weighted aggregation of inputs and biases at each node).
2. The prover sends the commitment to the DNN functions to the verifier.
3. The verifier sends the input data to the prover.
4. The prover sends the value of function evaluation for the DNN function at $layer_1$ and proof of such function evaluation to the verifier.
5. The verifier checks the correctness of such value of function evaluation using the DNN function commitment for $layer_1$ and proof of function evaluation for $layer_1$.
6. Next, the prover uses the value of function evaluations as inputs to $layer_2$ functions.
7. Next, steps 4 and 5 are repeated for all layers.

Setup for Common Reference String. During this step, the prover and the verifier compute a set of random numbers. Let G_1 be an elliptic curve for the prime number $\mathbb{P}$ and g_1 be the generator of G_1. Let τ be a random positive integer less than $\mathbb{P}$. A set of $l > n$ random numbers are generated from τ as follows:

$$\begin{aligned} p_1 = g_1, p_2 = M(\tau, g_1), p_3 = M(\tau^2(Mod(\mathbb{P})), g_1), \\ \dots, \dots \\ p_l = M(\tau^{l-1}(Mod(\mathbb{P})), g_1) \end{aligned} \tag{3}$$

where $M()$ is a function to multiply a positive integer with an elliptic curve point resulting in a new elliptic curve point. We can use a ceremony to form a random number. The prover and verifier know the above random points for both bilinear elliptic curves. We represent these random points for $\mathbb{G}_1, \mathbb{G}_2$ as follows:

$$\begin{aligned} \mathbb{G}_1 : & p_1^1 = g_1, p_2 = M(\tau, g_1), \dots, \\ & p_l^1 = M(\tau^{l-1}(Mod(\mathbb{P}_1)), g_1) \\ \mathbb{G}_2 : & p_1^1 = g_2, p_2 = M(\tau, g_2), \dots, \\ & p_l^2 = M(\tau^{l-1}(Mod(\mathbb{P}_2)), g_2) \end{aligned}$$

All parties are part of the Ethereum Blockchain network and smart contracts are used to execute the code for verifiers and aggregators.

Commitment to the DNN Functions. For the function $f(x) = y_1^1 = b_1^1 + \cup_{i=1}^n a_i w_{i,1}^1$, the prover creates a commitment as follows:

1. Let $F(x) = b_1^1 + w_{1,1}^1 x + w_{2,1}^1 x^2 + \dots + w_{n,1}^1 x^n$, i.e., coefficients of $F(x)$ is the set of weights and bias for a node in the DNN.
2. We will use KGZ polynomial commitment scheme to generate the commitment for $F(x)$.

$$\begin{aligned} Commit(F_1^1(\tau)) = & Add(Mul(b_1^1, \mathbb{P}_1), Mul(w_{1,1}^1, p_2^1), \\ & \dots, \\ & Mul(w_{n,1}^1, p_n^1)) \end{aligned} \tag{4}$$

where the Add function adds a set of elliptic curve points. Hence for each DNN layer, the prover creates n elliptic curve points for the elliptic curve $\mathbb{G}_1$, and in total, it creates nk such points. The set of commitments for layer i will be denoted as the set $\cup_{j=1}^n Commit(F_j^i(\tau))$.

Function Evaluation. Note that a DNN function $f(x) = y_1^1 = b_1^1 + \cup_{i=1}^n a_i w_{i,1}^1$ is evaluated as $b_1^1 + a_1 x^1 + a_2 x^2 + \dots + a_n x^n$. Hence we need to find a value for x that can represent the set of numbers $b_1^1, a_1, \dots, a_n$. The prover and the verifier generate a number x as follows:

$$
\begin{aligned}
x &= a_1 \\
x^2 &= a_2 \\
x^3 &= a_3 \\
&\cdots \\
x^n &= a_n \\
x + x^2 + x^3 + \cdots + x^n &= a_1 + a_2 + \cdots + a_n \\
x + x^2 + x^3 + \cdots + x^n - (a_1 + a_2 + \cdots + a_n) &= 0 \\
\frac{1}{1-x} &= (a_1 + a_2 + \cdots + a_n) \\
\frac{1}{(a_1 + a_2 + \cdots + a_n)} &= 1 - x \\
1 - \frac{1}{(a_1 + a_2 + \cdots + a_n)} &= x \qquad (5)
\end{aligned}
$$

The above equation holds for large n. We assume that the DNN model is large and hence n is large. Using this value of x, the prover calculates DNN functions for $layer_1$. We denote such a set of evaluations as the set $\{y_i^1\}$. Note that, the set $\{y_i^1\}$ is the input to $layer_2$, and for $layer_2$ we calculate the value of x for DNN functions of $layer_2$ using the same procedure.

Proof of Evaluation. Note that, value of evaluation for DNN functions $\{F_i^1(x)\}$ at $layer_1$ is the set $\{y_i^1\}$. For each function F_i^1, the prover creates a proof of evaluation as a point in the elliptic curve in $\mathbb{G}_1$ as follows: Let,

$$
\begin{aligned}
Q_1^1(x) &= \frac{F_1^1(x) - y_1^1}{x - a} \qquad (6) \\
&= r_0 + r_1 x + r_2 x^2 + \cdots + r_n x^n \\
Commit(Q_1^1(\tau)) &= Add(Mul(r_0, p_1^1), Mul(r_1, p_2^1), \\
\ldots, M(r_n, p_n^1)) \qquad (7)
\end{aligned}
$$

The set of proof of evaluations for $layer_1$ will be denoted as the set

$$\{Commit(Q_i^1(\tau))\}.$$

Verification of Proof of Evaluation. The verifier has the following values:

1. The set of commitment to the DNN functions at $layer_1$ as $\cup_{j=1}^{n} Commit(F_j^i(x))$.
2. The set of value of DNN function evaluation at $layer_1$ as $\{y_i^1\}$.
3. The set of commitment for proof of function evaluation to the DNN functions at $layer_1$ as $\{Commit(Q_i^1(\tau))\}$.

The verifier checks the validity of the commitment to the function evaluations by checking if the following equivalence holds:

$$\mathbb{E}(Commit(Q_i^1(\tau)), (\tau - x)) \equiv \mathbb{E}(Commit(F_j^i(x)) - y_i^1) \tag{8}$$

Note that the above equation checks if the commitment to the proof of function evaluation is correct, i.e., the polynomial division for Eq. 5 has no reminder. Note that verification requires multiplication of two elliptic curve points and hence we used bilinear pairing-based elliptic curves for this purpose.

Iteration for Complete DNN Execution Proof. After completing the procedure for proof of execution for $layer_1$, proof of execution for all remaining layers is also checked as follows:

1. Outcomes from $layer_1$ DNN functions, i.e., $\{y_i^1\}$ act as the input to $layer_2$.
2. x is calculated from $\{y_i^1\}$ using the method shown in Sect. 4.5.
3. The prover again generates the value for function evaluation and computes the proof of evaluation using the method shown in the previous section.
4. This process continues until the last layer. After verification of the last layer, the verifier can check if the value of the DNN function corresponds to the classification result given by the prover. Hence the verifier can verify the classification result.

Using the above DNN execution proof the verifier can form a binary model of the DNN functions where if the DNN function value is positive then it is regarded as 1 and otherwise it is regarded as 0. Hence the verifier forms a binary number from the DNN function evaluations.

Given the set of DNN function evaluation values and proof of DNN function evaluation proofs, the verifier can check if the binary representation of each DNN function is valid or not. We will use Solidity-based smart contracts for executing the DNN function verification as shown in the table below. It uses private values in smart contracts to prevent DNN function values from being revealed. After verification of the DNN models the aggregator generates a score for the parties as $\mathcal{S}_i$ such that

$$\mathcal{S}_i = A + B$$

where A is the number of test data with correct classification and B is the number of test data with incorrect classifications.

Commitment to Verified DNN Functions. After verification of DNN function evaluations by the verifier, each prover sends a commitment to its DNN functions as follows: If a DNN function has a set of weights and bias as $\{w_i\}$ and b_i then, commitment to this DNN function is the elliptic curve point $Mul(b_i + \sum w_i, g_1)$.

4.3 Coalition Formation Protocol

We will use the party-affiliation game [1] to model the coalition among the parties. In a party affiliation game parties are represented as an undirected graph where edges

are weighted and each party has either a friend or foe relation with other parties in its neighborhood defined by the graph. The objective of the party affiliation game is to partition the parties into groups (subgraphs) where the utility of a party is defined as

$$Utility = \sum_{\text{Friends in its group}} \text{Weights of edges} + \sum_{\text{Foes in its other group}} \text{Weights of edges} \tag{9}$$

Let $\mathcal{G}$ be an undirected connected graph among the parties (of the federated machine learning platform) with the set of edges $\mathcal{E}$ be the set of edges among the parties $\mathcal{U}$ be the set of edges. We define weight of the edge between two parties $\mathcal{U}_i$ and $\mathcal{U}_j$ as follows:

$$(\mathcal{U}_i, \mathcal{U}_j) \in Friend \qquad W(\mathcal{U}_i \rightarrow \mathcal{U}_j) = Abs(\mathcal{S}_i + \mathcal{S}_j)$$
$$(\mathcal{U}_i, \mathcal{U}_j) \in Foe \qquad W(\mathcal{U}_i \rightarrow \mathcal{U}_j) = -Abs(\mathcal{S}_i + \mathcal{S}_j)$$

The coalition formation algorithm generates subsets of the graph $\mathcal{G}$ such that in each subset the total weight of the edges is the summation of all edges in the subset. Let P^1 and P^2 be the partition of the parties into two groups such that parties in P^1 have accuracy more than the average accuracy of the parties and parties in P^2 have accuracy less than or equal to the average accuracy of the parties. We will say a pair of parties have a foe relation if both of them do not belong to either P^1 or P^2, otherwise, they have a friend relation. Let $\mathcal{H}_1, \ldots, \mathcal{H}_\S$ be a set of x subgraphs. Utility of a party $Party_y$ in the subgraph $\mathcal{H}_i$ is defined as follows:

$$Utility = \sum_{Party_x \in \mathcal{H}_i \wedge (Party_x, Party_y) \in Friend} W(Party_x, Party_y) + \sum_{Party_x \notin \mathcal{H}_i \wedge (Party_x, Party_y) \in Foe} W(Party_x, Party_y) \tag{10}$$

The coalition formation solution, i.e., the subgraphs $\mathcal{H}_1, \ldots, \mathcal{H}_\S$ is stable if no party can switch groups to improve its utility. We will use the algorithm below to generate these subgraphs as a solution to the coalition formation problem. It is as follows:

1. First a random partition over the is generated as a set of subgraphs $\pi_1, \ldots, \pi_x$ each with at most k parties.
2. R is a random order among the parties.
3. Each party gets a chance to move to another group if by doing so resulting coalition's weight increases.
4. Above two steps are repeated for a finite number of iterations.

4.4 Identify Subsets of Weights and Biases

We will the smart contract to evaluate compatibility among binary DNN models. Compatibility among two binary representation of two DNN models as follows(Table 2):

Algorithm 1. Coalition formation protocol.

Data: $\mathcal{G} = \mathcal{U}, \mathcal{E}, W(\mathcal{E})$
Result: Subgraphs of the graph as coaltions
begin
  $\Pi = (\pi_1, \ldots, \pi_x)$ subgraph initially with random x vertices
  for *Each* $i \in [1 : x]$ **do**
    if *Size of* π_i *less than* k **then**
      Randomly choose $u \notin \Pi$
      if $\exists v \in \pi_\rangle : (u, v) \in \mathcal{E}$ **then**
        Add v to π_i
  for $i \in [1 : n]$ **do**
    $W(\pi_i) \leftarrow \sum Utility(v_i \in \mathcal{V}(\pi_i))$
  R be a random order among the parties
  while *Repeat for* z *times* **do**
    for *Each party* $\mathcal{U}_i$ **do**
      π_x such that $\mathcal{U}_i \in \mathcal{V}(\pi_x)$
      $W(\pi_x(-\mathcal{U}_i)$ weight of the coalition without $\mathcal{U}_i$
      if $W(\pi_x(-\mathcal{U}_i) \geq W(\pi_x$ **then**
        $W(\pi_y(+\mathcal{U}_i)$ weight of the coalition adding $\mathcal{U}_i$
        if $W(\pi_y(+\mathcal{U}_i) > W(\pi_y)$ **then**
          if $W(\pi_y(+\mathcal{U}_i)$ *is maximum for all* $\Pi(-\pi_x)$ **then**
            Move $\mathcal{U}_i$ to π_y
  Return(Π)

1. We compute bit-wise XOR for binary DNN representation of two models.
2. Let $Index$ be the set of indices with And value as 1.
3. These two parties should exchange DNN parameters corresponding to the set of indices $Index$.

Table 2. Procedure for Binary DNN function comparison.

Party	x_1	x_2	...	x_{n-1}	x_n
$Party_x$	1	1	...	1	0
$Party_y$	1	0	...	0	1
XOR	0	1	...	1	1

4.5 Verified Model Exchange

The protocol for verified model exchange is as follows:

1. The party (verifier) who wants to receive subsets of DNN model parameters of another party (Prover), performs an oblivious transfer protocol to get the DNN

model parameters. Let the addition of elliptic curve points for these parameters be Q_2. This procedure is shown in Table 3.
2. Next, the prover sends the summation of elliptic curve points corresponding to the model parameters not sent to the verifier via oblivious transfer. Let this point be Q_1. This procedure is shown in Table 4.
3. Now the verifier checks if the following holds $Q_i = Q_1 + Q_2$

The oblivious transfer protocol for model parameter exchange is as follows:

1. In this protocol prover is the party who has to share a subset of its DNN model parameter with another party who is the verifier.
2. Prover generates a random number a and two elliptic curve points $A = Mul(a, g_1)$ and $T = Mul(a, A)$ and it sends A to the verifier.
3. Let $\{c_i\}$ be the index of chosen DNN parameters. The verifier generates a set of random numbers $\{b_i\}$ one for each index of the DNN parameter it wants to receive from the prover.
4. For each chosen index i of the DNN parameter it wants to receive, the verifier computes $\{M_i = Mul(c_i, A)\}$ and $\{N_i = Mul(b_i, g_1)\}$. Then it generates $\{R_i = Add(M_i, N_i)\}$ and sends $\{R_i\}$ to the prover.
5. The prover generates keys (one for each DNN parameter) as

$$K_i = Sub(Mul(a, R_i), Mul(i, T))$$

and encrypt i'th DNN parameter with k_i and sends the encrypted data to the verifier.
6. The verifier decrypts the data with the key $Mul(b_i, A)$. Note that:

$$\begin{aligned}
K_i &= Sub(Mul(a, R_i), Mul(i, T)) \\
&= Mul(a, R_i) - Mul(i, T) \\
&= Mul(a, R_i) - Mul(i, T) \\
&= Mul(a, (M_i + N_i)) - Mul(i, (a, A))) \\
&= Mul(a, (Mul(c_i, A) + Mul(b_i, g_1))) - Mul(i, (a, A))) \\
&= Mul(a * c_i, A) + Mul(a * b_i, g_1) - Mul(i, (a, A))) \\
&= Mul(a * c_i, A) + Mul(b_i, A) - Mul(i, (a, A))) \\
&= Mul(c_i, T) + Mul(b_i, A) - Mul(i, T)) \\
&= Mul(b_i, A) \qquad (11)
\end{aligned}$$

The oblivious transfer protocol [15] for verification of model parameter exchange is as follows:

1. It is similar to the oblivious transfer protocol used in transferring DNN parameters shown in Table 3.
2. For each DNN parameter, the prover generates an elliptic curve point by multiplying it with the generator of the elliptic curve.

Table 3. Procedure for Verified DNN model exchange.

Prover	Verifier
	For the DNN function F_j^i be $\{Id_x\}$ be the indices of the coefficient that the verifer wants from the Prover
Generate a random positive integer less than $\mathbb{P}_1$	
Compute $A = Mul(a, \mathbb{G}_1)$	
Compute $T = Mul(a, A)$	
Send A	
	$\{Id_x\} \in [1 : n]$ be the chosen DNN model parameters indices
	Generate $C = c_1 \ldots, c_x$ (x is is number of chosen indices)
	Generate $B = b_1 \ldots, b_x$ (x is is number of chosen indices)
	Compute $\{M_i = (Mul(c_i, A)\}$ $\{N_i = Mul(b_i, g_1))\}$
	Compute $\{R_i = Add(M_i, N_i)\}$
	If $i \notin \{Id_x\}$, R_i is a random elliptic curve point
	Send $\{R_i\}$ to Prover
Generate n keys as $K_i = Sub(Mul(a, R_i), Mul(i, T))$ Encrypt ith DNN function data with k_i and send it to the verifier	
	Decrypt i'th data with the key $Mul(b_i, A)$

3. After execution of the protocol the verifier generates two elliptic curve points

$$\begin{aligned} Q_1 &= Add \text{ elliptic curve point for chosen DNN parameters} \\ Q_2 &= Add \text{ elliptic curve point for non-chosen DNN parameters} \end{aligned} \tag{12}$$

4. Verfier gathers the commitment to the DNN parameter Q_i as the addition of elliptic curve points for all DNN parameters as shown in Table 4 and it checks if the following holds:

$$Q_i = Add(Q_1, Q_2)$$

4.6 Summary of All Federated Machine Learning Procedures

A summary of all federated machine learning procedures is shown in the figure below. It is as follows:

1. After the generation of local DNN models, the aggregator sends a test data set to evaluate the local DNN models.
2. Each party sends the classification results, proof of evaluation, and proof of binary DNN to the aggregator.

Table 4. Procedure for Verified DNN model exchange.

Prover	Verifier
	For the DNN function F_j^i be $\{Id_x\}$ be the indices of the coefficient that the verifer wants from the Prover
Compute $\{q_i = Mul(p_i, g_1)\}$ for each DNN model parameter	
Generate a random positive integer less than $\mathbb{P}_1$	
Compute $A = Mul(a, \mathbb{G}_1)$	
Compute $T = Mul(a, A)$	
Send A	
	$\{Id_x\} \in [1 : n]$ be the chosen DNN model parameters indices
	Generate $C = c_1 \ldots, c_x$ (x is is number of chosen indices)
	Generate $B = b_1 \ldots, b_x$ (x is is number of chosen indices)
	Compute $\{M_i = (Mul(c_i, A)\}$ $\{N_i = Mul(b_i, g_1))\}$
	Compute $\{R_i = Add(M_i, N_i)\}$
	If $i \notin \{Id_x\}$, R_i is a random elliptic curve point
	Send $\{R_i\}$ to Prover
Generate n keys as $K_i = Sub(Mul(a, R_i)$, $Mul(i, T))$ Encrypt ith DNN function data with k_i and send it to the verifier	
	Decrypt i'th data with the key $Mul(b_i, A)$
	Add all decrypted elliptic curve points $Q_1 = sum(\{q_i\})$
	Compute elliptic curve points for all DNN model parameters in $\{Id_x\}$ and add these points as Q_2
	Check $Q_i = Q_1 + Q_2$

3. If the aggregator can verify the proof of evaluation and the same for the binary DNN then it will ask each party to submit a commitment to its DNN functions.
4. Using the accuracy of test data classification results the aggregator generates the coalition structure and it informs the parties about the coalitions and indices of weights and biases that they should exchange.
5. Parties with each coalition exchange DNN model parameters using oblivious transfer protocol and each party verifies the DNN model parameters of another party.

5 Analysis

Theorem 1. *Nash equilibrium exists for the coalition formation game.*

Proof. The coalition formation game defined using the utility function shown in Eq. 10 is a potential game. A potential game always has Nash equilibrium [1].

Theorem 2. *Trust problems 1,2,3 are resolved.*

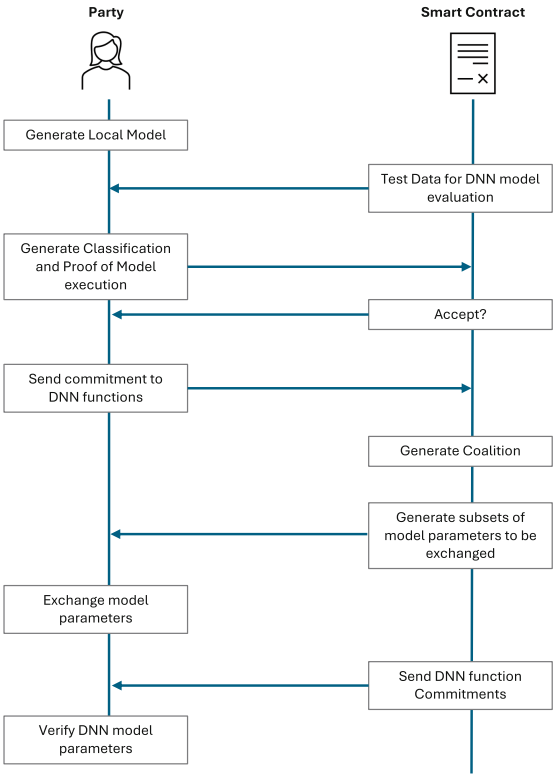

Fig. 4. Summary of all procedures for the coalition-based federated machine learning.

Proof. (Trust problem 1): In Trust problem 1, we need to trust a party for correct DNN model execution. A malicious party may report random or wrong classification for the test data given by the aggregator. The zkSNARK-based verification shown in Sect. 4.2 ensures that (a) the classification result is due to the execution of the DNN model of a party, and (2) the binary DNN model is due to the correct execution of the DNN model of a party.

(Trust problem 2): In Trust problem 2, we need to trust the aggregator for correctly executing the coalition formation algorithm. We eliminate such trust requirements by executing the coalition formation algorithm in a smart contract. Each party can review the smart contract code and as binary DNN models are verified by zero-knowledge proof, we do not need to trust the aggregator.

(Trust problem 3): In Trust problem 3, a party needs to trust another party for correct DNN model parameters. We prevent a party from reporting incorrect DNN model parameters because:

1. We use oblivious transfer protocol to share DNN model parameters. As a party does not know which DNN model parameter it is reporting to another party, it can not correctly choose the parameters to be modified.

2. Each party keeps a commitment to its DNN model as shown in Sect. 4.2 and using oblivious transfer protocol one party generated two commitments from another party (who is reporting its DNN model) as (a) commitment to the reported parameters and (b) commitment to the unreported parameters. As the reporting party does not know the parameters it is reporting it can not manipulate such commitments so that these commitments are valid w.r.t to the commitment generated for the entire DNN function.

6 Evaluation

We will use the MNIST dataset (https://keras.io/api/datasets/mnist/) for evaluating the secure coalition formation for federated machine learning. We use 30 parties and partition the dataset into 30 equal-size training sets. We use a random connected undirected graph among the parties with an average degree of 3. We find the initial partition among the parties by generating three random subgraphs with 10 parties in each group. Next, we use the following procedure to modify these groups:

1. We evaluate the average accuracy of each group using test data from the MNIST dataset.
2. Next, using the coalition formation algorithm we generate a new coalition among the parties.
3. We again evaluate the accuracy of the modified coalitions.
4. We repeat this process for 4 times.

A summary of the experimental evaluations is as follows:

1. As shown in Fig. 5, the average accuracy of the parties improves as coalitions are gradually changed. This shows that the coalition-based federated machine learning method improves coalitions with each iteration.
2. As shown in Fig. 6, the number of parties with accuracy less than the average accuracy of the parties decreases as coalitions are gradually changed. It shows that the parties with a less accurate DNN model benefited from the coalition formation algorithm as the number of parties with a less accurate DNN model decreases as we gradually modify the coalition with each iteration.
3. As shown in Fig. 7, the average accuracy of the parties who have less than average accuracy for the first coalition improves as coalitions are gradually changed. It shows that parties with initially less accurate DNN model improve their DNN model with each iteration. Hence they have no incentive to leave the coalition.

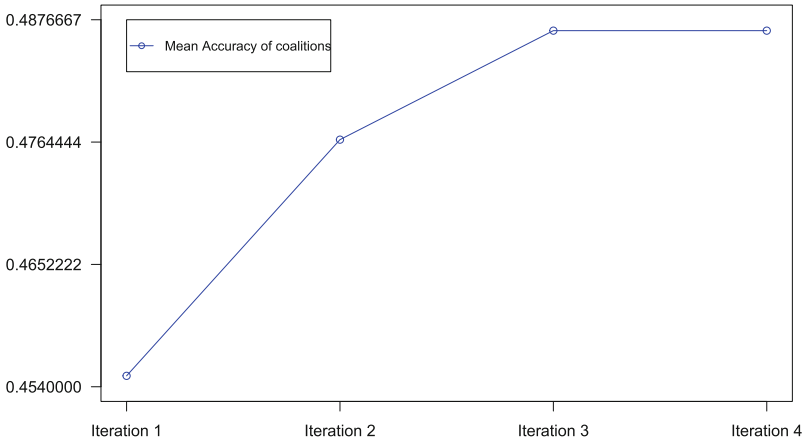

Fig. 5. Average accuracy of the parties as coalitions are gradually changed.

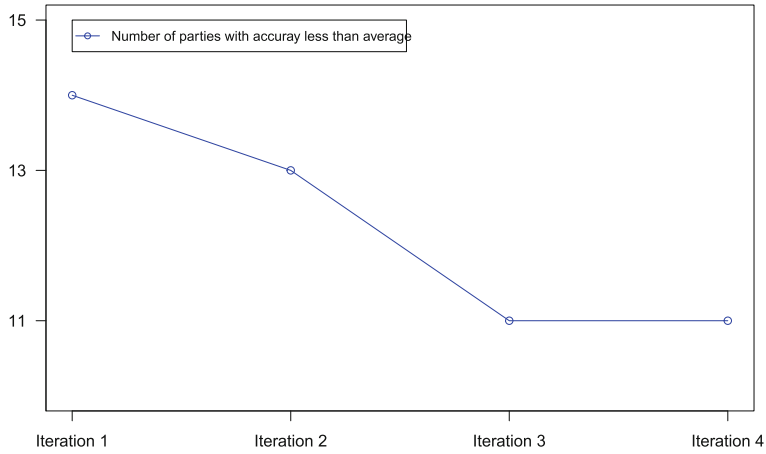

Fig. 6. Number of parties with accuracy less than average accuracy as coalitions are gradually changed.

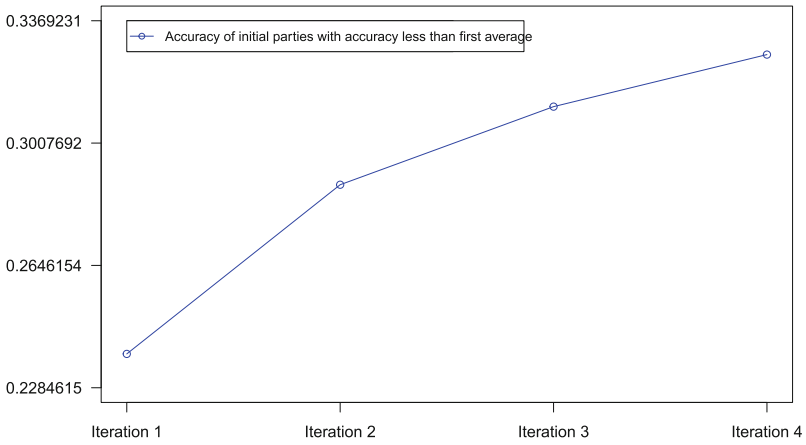

Fig. 7. Average accuracy of parties with accuracy less than initial average accuracy.

7 Conclusion

In this paper, we proposed a coalition formation procedure for federated machine learning. We solve multiple trust issues for such a procedure including verification of DNN model execution, correct coalition formation algorithm execution, and verification of correct DNN model sharing. Additionally, we prove that the proposed coalition formation protocol is stable. In the future, we will explore use cases of coalition formation-based federated machine learning.

Acknowledgements. This publication has emanated from research supported by grants from Science Foundation Ireland (SFI) under Grant Numbers 21/FFP-A/9174 (Sustain), 16/RC/3835 (VistaMilk) and 12/RC/2289_P2 (Insight). For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

1. Bhalgat, A., Chakraborty, T., Khanna, S.: Approximating pure nash equilibrium in cut, party affiliation, and satisfiability games. In: Proceedings of the 11th ACM Conference on Electronic Commerce. EC 2010, Association for Computing Machinery, New York, NY, USA, pp. 73–82 (2010). https://doi.org/10.1145/1807342.1807353
2. Bonawitz, K.A., et al.: Practical secure aggregation for federated learning on user-held data. In: NIPS Workshop on Private Multi-Party Machine Learning (2016). https://arxiv.org/abs/1611.04482
3. Elkordy, A.R., Avestimehr, A.S.: Secure aggregation with heterogeneous quantization in federated learning. CoRR arXiv:2009.14388 (2020)
4. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12

5. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
6. Hamer, J., Mohri, M., Suresh, A.T.: FedBoost: a communication-efficient algorithm for federated learning. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 3973–3983. PMLR (2020). https://proceedings.mlr.press/v119/hamer20a.html
7. Jeon, B., Ferdous, S.M., Rahman, M.R., Walid, A.: Privacy-preserving decentralized aggregation for federated learning. CoRR arXiv:2012.07183 (2020)
8. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
9. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: challenges, methods, and future directions. IEEE Signal Process. Mag. **37**, 50–60 (2019). https://api.semanticscholar.org/CorpusID:201126242
10. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8269, pp. 41–60. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42033-7_3
11. Liu, L., Zhang, J., Song, S., Letaief, K.B.: Hierarchical federated learning with quantization: convergence analysis and system design. IEEE Trans. Wireless Commun. **22**(1), 2–18 (2023). https://doi.org/10.1109/TWC.2022.3190512
12. Lyu, L., Yu, H., Yang, Q.: Threats to federated learning: a survey. CoRR arXiv:2003.02133 (2020)
13. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.Y.: Communication-efficient learning of deep networks from decentralized data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (2017). https://proceedings.mlr.press/v54/mcmahan17a.html
14. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252 (2013). https://doi.org/10.1109/SP.2013.47
15. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. **2005**, 187 (2005). https://api.semanticscholar.org/CorpusID:15222660
16. Reddi, S., et al.: Adaptive federated optimization. arXiv:2003.00295 (2021)
17. Savazzi, S., Nicoli, M., Rampa, V., Kianoush, S.: Federated learning with mutually cooperating devices: a consensus approach towards server-less model optimization. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3937–3941 (2020). https://doi.org/10.1109/ICASSP40776.2020.9054055
18. Segal, A., et al.: Practical secure aggregation for privacy-preserving machine learning. In: CCS (2017). https://eprint.iacr.org/2017/281.pdf
19. Tang, X., Yu, H.: Competitive-cooperative multi-agent reinforcement learning for auction-based federated learning. In: Elkind, E. (ed.) Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, pp. 4262–4270. International Joint Conferences on Artificial Intelligence Organization (2023). https://doi.org/10.24963/ijcai.2023/474
20. Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D.S., Khazaeni, Y.: Federated learning with matched averaging. CoRR arXiv:2002.06440 (2020)
21. Wei, K., et al.: Federated learning with differential privacy: algorithms and performance analysis. arXiv:1911.00222 (2019)

22. Xiang, W.T., Shao, M., Fu, Y., Jia, R., Lin, F., Zheng, Z.: Federated learning framework based on trimmed mean aggregation rules (2022). https://openreview.net/forum?id=AUszBTiYBB6
23. Zhao, L., Jiang, J., Feng, B., Wang, Q., Shen, C., Li, Q.: Sear: secure and efficient aggregation for byzantine-robust federated learning. IEEE Trans. Dependable Secure Comput. **19**(05), 3329–3342 (2022). https://doi.org/10.1109/TDSC.2021.3093711