Peer-Reviewed Federated Learning

Mattia Passeri[1], Andrea Agiollo[1,*] and Andrea Omicini[1]

[1]Dipartimento di Informatica – Scienza e Ingegneria (DISI), Alma Mater Studiorum—Università di Bologna, Italy

Abstract

While representing the de-facto framework for enabling distributed training of Machine Learning models, Federated Learning (FL) still suffers convergence issues when non-Independent and Identically Distributed (non-IID) data are considered. In this context, the local model optimisation on different data distributions generate dissimilar updates, which are difficult to aggregate and translate into sub-optimal convergence. To tackle this issues, we propose *Peer-Reviewed Federated Learning* (PRFL), an extension of the traditional FL training process inspired by the peer-review procedure common in the academic field, where model updates are reviewed by several other clients in the federation before being aggregated at the server-side. PRFL aims at enabling the identification of relevant updates, while disregarding the ineffective ones. We implement PRFL on top of the Flower FL library, and make *Peer-Reviewed Flower* a publicly-available library for the modular implementation of any review-based FL algorithm. A preliminary case study on both regression and classification tasks highlights the potential of PRFL, showcasing how the distributed solution can achieve performance similar to that obtained by the corresponding centralised algorithm, even when non-IID data are considered.

Keywords

Federated Learning, non-IID data, Peer-review

1. Introduction

Federated Learning (FL) [1, 2] represents the de-facto framework for enabling distributed training of Machine Learning (ML) and Deep Learning (DL) models. In FL scenarios, a central server interacts with multiple users – also called clients or workers – to train a ML/DL model jointly. Each client locally trains its ML model on its private data, while the server aggregates local updates upon their reception. The clients never share local raw data, instead they propagate updates from the training process over those data. Therefore, the groundbreaking idea of FL is for the training process to take into account all clients data while never disclosing their nature, thus maintaining their privacy [3, 4]. Distributing the computation over all clients FL also achieves efficiency improvements over centralised training approaches, making FL more popular than centralised approaches whenever large systems are taken into account [5].

While being so popular, FL does not represent a silver-bullet solution for enabling distributed training of ML and DL models. One of the most severe shortcomings of FL systems is due to the locality of model training, with most FL approaches suffering from non-Independent and

WOA 2023: 24th Workshop From Objects to Agents, November 6–8, Rome, Italy

*Corresponding author.

mattia.passeri2@studio.unibo.it (M. Passeri); andrea.agiollo@unibo.it (A. Agiollo); andrea.omicini@unibo.it (A. Omicini)

0000-0003-0531-1978 (A. Agiollo); 0000-0002-6655-3869 (A. Omicini)

Identically Distributed (non-IID) data issues [6]. In this context, several works have shown how variability of data distribution, label distribution, and data quality among the clients belonging to the federation can hinder the optimisation process of FL systems [7, 8, 9]. Conceptually speaking, the optimisation of several local models on different data distributions generates very different updates, which are hard to aggregate at server-level and translate into sub-optimal global model updates.

Inspired by such limitations, in this paper we introduce Peer-Reviewed Federated Learning (PRFL) as an extension of the traditional FL process inspired by the peer-review procedure, common in the academic field. PRFL relies on the addition of a validation phase of local updates produced by each of the clients involved in the federation process. In particular, PRFL splits each iteration of federated training in two phases, namely the *training* phase and the *review* phase. The training phase corresponds to a single round of traditional FL training. On the other hand, the review phase corresponds to multiple rounds of communication between the server and the clients. During this phase the provisional updates produced by each client in the federation are subjected to one or more validation cycles by other clients, aiming at evaluating how good the update is for the whole federation. At the end of the review phase, the server aggregates the resulting provisional updates, depending on the reviews feedback, to obtain the global round update. The aim of the review phase is the identification of the most valuable updates to be considered for server-side aggregation, simplifying the procedure when non-IID data are considered.

The proposed PRFL approach is implemented on top of the popular Flower FL library [10], enabling the deployment and simulation of peer-reviewed federated learning approaches. A simple case study is considered to test the performance of PRFL, tackling both classification and regression tasks, defining a novel distributed training approach for Gradient Boosted Decision Trees (GBDT) models, which requires revision. The proposed approach achieves similar performance to the centralised training model, even when non-IID data distributions are considered.

To summarise, the contribution of our paper are the following:

- we propose a novel peer-review based approach for the federated learning paradigm, aiming at tackling the non-IID data issues of FL systems;
- we implement Peer-Review Flower (PRFlower)[1], a Flower-based library implementing PRFL that is modular and flexible enough to allow for the implementation of almost any review-based FL approach;
- we test the proposed review-based approach on regression and classification tasks via the proposal of Federated Least Squares Boosted Trees (FedLSBT) as an extension of the popular Gradient Boosted Decision Trees (GBDT) models.

[1]https://github.com/passerim/peer-review-flower

2. Background

2.1. Federated Learning

Most, if not all, FL scenarios consider a central server interacting with multiple clients to jointly train a shared ML model. Throughout the reminder of this paper we will refer to the central server simply as server or Central Aggregator (CA), interchangeably. In this context, we consider a federation network made of N clients. Each worker locally trains its ML model on its own private data for a predefined period of time—namely training iterations. Thus, each federation client obtains a local update – depending on its available data – over the global ML model. The FL training procedure relies on clients periodically sending the results of their local training – under the form of whole model [11, 12], obtained gradient [13, 14], or other possible solutions – to the server, which is then in charge of aggregating the local updates to compute the joint global update. Several aggregation solutions have been proposed recently – such as FedAvg [13], Newton-type methods [15], etc. –, aiming at maximising the effectiveness of the federated training procedure over different scenarios. Once the global update is available, the server needs to propagate it to all the federation clients, ensuring synchronisation between clients and the global federation state. Formally, the objective of FL is to cooperatively find a global model M that minimises a finite-sum objective function of the form

$$\min_{M \in \mathbb{R}^d} f(M) \quad \text{where} \quad f(M) = \frac{1}{D} \sum_{i=1}^{P} D_i f^{(i)}(M) \tag{1}$$

where D is the overall number of data samples of the global FL problem, while D_i and $f^{(i)}(M)$ are the number of data samples and the local cost function of the i-th client, respectively. The global optimisation process is repeated T times aiming at achieving model convergence.

2.2. Federated Learning Frameworks

Currently, a number of open source frameworks support the implementation of federated learning algorithms, namely:

- *FedML* [16]: a FL framework based on PyTorch[2]. Its architecture includes a core layer that contains training logic and implements distributed communication protocols, while a high-level API allows implementation of various federated learning algorithms.
- *PySift* [17]: also based on PyTorch, it focuses on distributed *secure* computation and differential privacy.
- *Tensorflow Federated*[3]: developed by Google as an extension of Tensorflow, its architecture is divided into two levels: a core level which gives the possibility to carry out federated computations and a higher level API for working with models, datasets and implementing algorithms. This framework can only be used to simulate federated learning systems as it does not support distributed real-world deployment.

[2]https://pytorch.org
[3]https://www.tensorflow.org/federated

- *Flower* [10]: agnostic with respect to the framework chosen to implement the predictive models, *Flower* allows both distributed and local deployment, in the form of simulations, of federated learning systems on heterogeneous devices, with client-side implementations of the library also available for mobile platforms (iOS, Android, and IoT).

Among the available alternatives, we adopt *Flower* as the basis for the PRFL framework implementation, since it is easily extendable, applicable to both distributed contexts and simulations, and agnostic w.r.t. the underlying ML library used.

2.2.1. Flower's Architecture

Two sorts of computations exist in FL, namely: *(i)* computations performed locally by clients having access to data; and *(ii)* computations performed by the server to aggregate the clients' computations results. Flower's architecture reflects this feature and breaks the algorithmic logic into two parts, one executed on the clients and one executed on the server. The *server-side* logic manages through a series of calls of the *Strategy* class *(i)* the clients selection process, *(ii)* the configuration of both training and validation of the global model, and *(iii)* the procedure to aggregate the results obtained by the clients. Another fundamental component on the server side is the *ClientManager* class, which manages a set of *ClientProxy* objects, each of which represents a client connected to the server and allows the server to exchange messages with the client itself. Meanwhile, the *client-side* logic is purely reactive, where clients – which are instances of the class *Client* – await the reception of messages from the server and execute the handler associated with the type of message received. The types of messages provided by the Flower protocol and their handlers are:

- *GetPropertiesIns*: it contains a key-value dictionary. Its reception triggers the execution of the get properties callback. It can be used to send and receive information between client and server.

- *GetParametersIns*: its reception triggers the execution of the get parameters callback whose result is a message containing the parameters of the client model.

- *FitIns*: this message carries the global model sent from the server to the client and a training configuration dictionary. Upon the message reception the client executes the fit callback whose result must be a message containing: the updated model, the number of samples used for training and a dictionary containing any other information needed to be returned to the server.

- *EvalIns*: is the message with which the server sends the global model to the client to validate it on the local test set. Upon the message reception the client executes the evaluate callback which returns a message containing the desired validation metrics.

Figure 1 shows Flower's architecture in a simulated environment, highlighting the dependencies amongst the server-side and client-side components.

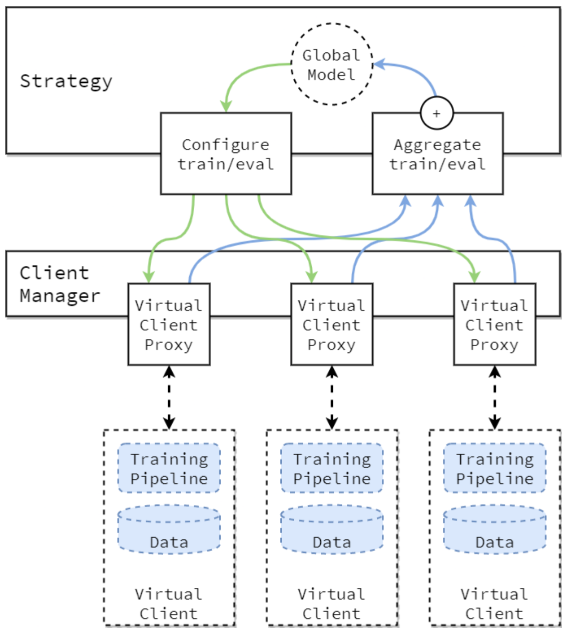

Figure 1: Flower's architecture in a simulated environment.

3. Peer-Reviewed Federated Learning

We introduce Peer-Reviewed Federated Learning (PRFL) as an extension of the traditional FL process. PRFL relies on the addition of a validation phase of local updates produced by each of the clients involved in the federation process. Therefore, the optimisation procedure in PRFL is split in two phases: *(i)* the *training* phase, and *(ii)* the *review* phase. The training phase corresponds to the traditional FL training round in which a given number of clients are selected to produce a set of provisional updates. On the other hand, the review phase corresponds to multiple rounds of communication from the server to the clients and vice-versa. During this phase the provisional updates produced by each client in the federation go under one or more validation cycles by other clients, aiming at evaluating the update goodness for the whole federation. In this context, a single review cycle includes:

1. The selection by the server of a subset of the clients that participate in the review round. The number of clients M selected for the review round represents an hyperparameter of the proposed approach, which should be ideally tuned depending on the scenario at hand. A higher number of reviewers M would ideally result in a more meaningful review process, as they would cover a greater portion of the data available in the federation, but it would also introduce a higher computational burden and communication overhead. On the other hand, a smaller M makes the review process quicker, but less reliable since only a small portion of federation data are considered.

2. The distribution by the server to the selected clients of one or more temporary updates

of the global model, possibly together with the global model itself.

3. The validation by the M clients participating in the review round of the updates received from the other clients.

4. The server-side aggregation phase of the results returned by the clients after the review process, and possibly the production of a new set of provisional updates to the global model.

At the end of the review phase, the server aggregates the resulting provisional updates to obtain a new set of global model parameters to replace the current model parameters. The novel global model parameters can then be re-distributed amongst clients – as in traditional FL setups –, and the next optimization round can start. Algorithm 1 shows the pseudocode of the training process defined for PRFL.

Algorithm 1 PRFL training

```
Input:  T, M, n-iterations, review-rounds
Output:  trained-global-model
 1: send global model to all federation clients
 2: for n = 1 . . . n-iterations do
 3:     select T clients
 4:     send training instructions to selected clients
 5:     receive model updates and aggregate them
 6:     for m = 1 . . . review-rounds do
 7:         select M clients
 8:         send global model and candidate model to M clients
 9:         receive parameters reviews and aggregate them
10:         compute stop-review
11:         if stop-review then
12:             exit review
13:         end if
14:     end for
15:     aggregate candidate models parameters
16:     send global model to all federation clients
17: end for
18: trained-global-model = global model
```

The proposed algorithm is fully generic, and the aggregation procedure – which is dependent on the reviews received – is kept unspecified on purpose. Depending on the context at hand, different review-based aggregation procedures may perform differently: therefore we consider the aggregation policy to be a hyperparameter of our approach, which should be tuned context-dependently. Throughout our experiments we consider an extension of the FedAvg algorithm which at each iteration exploits the review phase to carry out a line search useful for determining the optimal extent of updating the parameters of the global model. More in detail, the global model update phase is carried out as the following: at the end of each training round, the individual updates produced by the clients are aggregated using FedAvg, however – unlike in common FedAvg – the update produced is not directly incorporated into the global model but is added to it after having been scaled by a quantity determined through a line search procedure performed by the clients selected for the review phase. Therefore, the review phase is useful to enable the line search procedure which identifies the optimal scaling quantity for the model update to be computed. Here it is worth noticing that the selected review process represents a very simple approach and is not guaranteed to achieve the optimal solution.

3.1. Implementation

We implement PRFL as a Python library extension of Flower. We refer to the library implementation of PRFL as Peer-Reviewed Flower (PRFlower), which is made publicly available[4] to ease future use and extension. PRFlower relies on the Flower FL library, and as such the components created are designed as specialisations of Flower components described in Section 2.2.1. We consider this setup to maintain maximum compatibility both in terms of interfaces and in terms of the use of components that do not need to be modified. The focus of our Flower extension (PRFlower) are the components that model the server, the clients and the learning strategy, following the reference learning process described in Algorithm 1. Such an approach makes it possible to avoid defining and modelling completely new entities, while requiring to appropriately specialise some of the Flower existing entities.

- a *PeerReviewServer* server, extending the original Flower's *Server* class. The new server class must enable the execution of multiple review rounds, the number of which might be established dynamically—i.e., upon the satisfaction of a pre-defined criterion.
- A *PeerReviewClient* client – extending Flower's *Client* class – which, upon the reception of specific messages must enable the execution of a review procedure of the model(s) received from the server, in addition to the original procedures for training and validation of the predictive model on the local dataset. Our design allows for the review process to be completely custombisable by the library user, allowing for an additional range of freedom.
- A *PeerReviewNumPyClient* client supporting the review process and being equivalent to the Flower's *NumPyClient* class—i.e. it offers immediate support for working with multidimensional array lists as a representation format for a predictive model.
- A strategy class *PeerReviewStrategy* that extends the callbacks provided by Flower's *Strategy* by adding appropriate methods to: *(i)* configure a review round and aggregate the results in a similar way to what Flower considers for the training round; *(ii)* aggregate – at the end of the review phase – the results obtained during the review rounds carried out; and *(iii)* verify – at the end of each review round – whether to continue with another review round or end the current iteration of the server learning cycle.
- A set of appropriate *TrainIns*/*TrainRes* and *ReviewIns*/*ReviewRes* messages that trigger clients to execute a training or review round. A *ReviewIns* message should carry the

[4]https://github.com/passerim/peer-review-flower

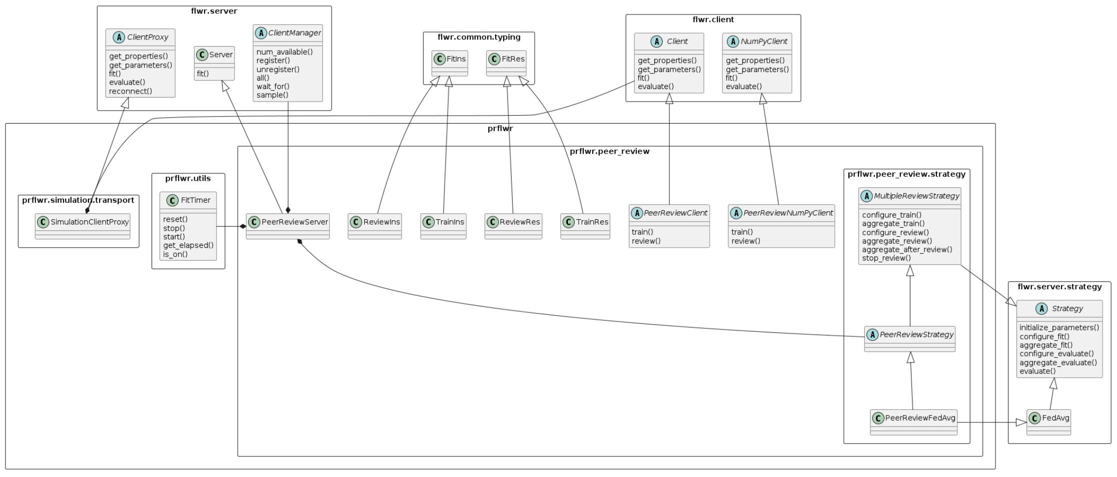

Figure 2: UML class diagram representing the main components of Peer-Reviewed Flower.

parameters of one or more global model updates in binary format and a configuration dictionary, in this way it is possible to provide clients with all the information they need for the review. Similarly, a *ReviewRes* message must be able to carry from the clients to the server a new update for the global model if the algorithm requires it and a dictionary containing various metrics. Both of these requirements are satisfied by the original *FitIns*/*FitRes* type messages. Therefore, in our PRFlower implementation we consider defining the *ReviewIns* and *ReviewRes* messages as subtypes of *FitIns* and *FitRes* respectively, adding in their configuration dictionary a *REVIEW FLAG* key-value pair. The added key acts as a flag to indicate whether the message is used for review – i.e., *REVIEW FLAG* $= 1$ – or training—i.e., *REVIEW FLAG* $= 0$. Therefore, a new pair of *TrainIns* and *TrainRes* messages is defined inheriting respectively from *FitIns* and *FitRes* and setting *REVIEW FLAG* $= 0$. This implementation choice allows PRFlower to use the original Flower protocol for the exchange of *FitIns* and *FitRes* messages between clients and servers. Upon its reception, each message will undergo routing in the clients and server towards the appropriate handlers depending on the value of the *REVIEW FLAG* field in the configuration dictionary.

Figure 2 summarises the novel components of the implementation of our PRFlower mechanism under an UML class diagram, highlighting the dependency between the novel peer review related classes and the original Flower implementation.

Thanks to its modular design and the advantages of Flower's generality, the proposed PRFlower architecture achieves:

- *flexibility*: the underlying training, review, and aggregation mechanisms are not specified in the PRFlower implementation, and as such can be specified by the library user, allowing for the definition of any review-based FL algorithm;
- *simplicity*: throughout the PRFlower design we ensure that the library allows users to

implement new algorithms by focusing predominantly on algorithmic logic, ignoring – possibly complex – aspects related to distribution and communication protocols;

- *simulation support*: extending the Flower library, PRFlower inherits the support for federation simulation, so that the implemented algorithms can be tested in simple contexts to evaluate their performance quickly before being deployed on real-world scenarios;
- *support for distributed deployment*: extending the Flower library, PRFlower also ensures that algorithms developed and tested in simulated contexts can be transferred to distributed environments over real-world scenarios, without much effort.

3.1.1. Usage

The distributed deployment of a peer-reviewed federated learning system using PRFlower involves a set of client nodes and a single server node. A client is a node with a Python environment running an instance of a class, implemented by the library user, that extends *PeerReviewClient*. Meanwhile, the server is a node provided with a Python environment in which it runs an instance of the *PeerReviewServer* class. In a distributed environment, the server communicates with clients via gRPC using the protocol and messages provided by Flower. Finally, the server needs to be provided with the implementation of a federated training strategy by extending the *PeerReviewStrategy* class, which implements the callbacks needed to configure the training, review, and validation phase of the model to be trained, and which is available to the *PeerReviewServer* class.

4. Case Study

In this section we showcase the possibilities provided by the PRFlower library and the corresponding PRFL algorithm. As an experimental application of this approach we consider the implementation of federated training of Gradient Boosted Decision Trees (GBDT) models. We select this sort of models as they represent the state-of-the-art solution for ML application to tabular data, and since there still exists a wide gap on the implementation of federated learning solutions for these models. Few approaches have been proposed in the literature, however their adoption is not really widespread as they show performance-related issues.

4.1. Gradient Boosted Decision Trees (GBDT)

Gradient boosting is a ML algorithm used for both regression and classification tasks that builds a predictive model as an ensemble of simpler models called weak learners. Typically, the choice of these models falls on decision trees—thus the name Gradient Boosted Decision Trees (GBDT). A GBDT model is built with an iterative procedure that combines at each step one or more trained trees in such a way to allow the optimisation of arbitrary objective functions as long as they are differentiable [18]. More in detail, the model learned by a GBDT algorithm has the following form:

$$F(x) = \gamma_0 + \sum_{m=1}^{M} \gamma_m h_m(x) \tag{2}$$

where γ_0 is a constant, while $h_m(x)$ is a decision tree and its contribution to the model prediction is weighted by the constant γ_m. To train a GBDT model, given a training set $D = (x_i, y_i), i = 1 \ldots N$ and a differentiable objective function $L(y, F(x))$, at each iteration $m = 1 \ldots M$ we build an $F_m(x)$ model with a greedy approach:

$$\begin{aligned} F_0(x) &= \gamma_0 = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma) \\ F_m(x) &= F_{m-1}(x) + \underset{h}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + h(x_i)) \end{aligned} \tag{3}$$

Approximating to the first order the objective function we obtain:

$$\begin{aligned} h_m(x) &= \underset{h}{\operatorname{argmin}} \sum_{i=1}^{N} h(x) \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \\ F_m(x) &= F_{m-1}(x) - \gamma_m h_m(x) \end{aligned} \tag{4}$$

therefore, up to a multiplicative constant:

$$h_m(x_i) \simeq -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \tag{5}$$

Therefore, at each iteration the gradients of all the samples in the training set are calculated with respect to the objective function and the model $h_m(x)$ is trained to predict the opposite of these values. In this way, the algorithm performs a sort of gradient descent where the constant γ_m represents the learning rate.

4.2. Federated Least Squares Boosted Trees (FedLSBT)

Starting from the original formulation of the gradient boosting algorithm, we develop Federated Least Squares Boosted Trees (FedLSBT), suitable for application in a federated context, and requiring the review phase of global model updates produced by clients. This algorithm enables training a global GBDT model shared by a set of clients without them exchanging data during training. Decision trees trained with a randomised procedure, called Extremely Randomized Trees (ExtraTrees) [19], are used as weak learners. We select ExtraTrees since the randomised choice of feature values on which to carry out node splits further protect client data privacy.

Let us consider a scenario in which there are K clients, a dataset D partitioned into K partitions D_k each stored in a different client and a differentiable objective function$L(y, F(x))$. In this context, the objective of federated learning is to train a global model $F(x)$ by optimising a function defined as the average of the objective function evaluated over all clients:

$$F^*(x) = \underset{F}{\operatorname{argmin}} \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|D_k|} \sum_{(x_i, y_i) \in D_k} L(y_i, F(x_i)) \tag{6}$$

Like GBDT, FedLSBT also builds the predictive model $F(x)$ as an additive ensemble of T trees $h_t(x)$ in which the contribution of each tree is weighted by a constant γ_t:

$$F(x) = \sum_{t=1}^{T} \gamma_t h_t(x) \tag{7}$$

At each iteration $m = 1, \ldots, M$ of federated training, a training round is performed in which an S_T subset of clients is sampled and a single review round in which an S_R subset of clients is involved – it is possible to take $S_R = S_T$ or perform a new sampling of clients. The idea behind the algorithm is to add $|S_T|$ trees to the global model for each training cycle learned during the training round, each weighted by a constant determined during the review phase.

$$F_m(x) = F_{m-1}(x) + \sum_{k=1}^{|S_T|} \gamma_k h_k(x) \tag{8}$$

Each client k involved in the training round trains a tree $h_k(x)$ on the local training set to predict the opposite of the gradient of the objective function with respect to the prediction of the global model and sends the update thus produced to the server. The updates received from the server at the end of the training phase are then used in the review phase to approximate the gradients of the objective function with respect to the prediction of the global model considering all the samples present in the private $D_{k'}$ datasets of the clients selected for the review, solving the following least squares problem with respect to the value of the constants γ_k:

$$\min_{\gamma_1, \gamma_2, \ldots, \gamma_{|S_T|}} \left[\sum_{(x,y) \in D_R} \left(\frac{\partial L(y_i, F_{m-1}(x))}{\partial F_{m-1}(x)} - \frac{1}{|S_T|} \sum_{k=1}^{|S_T|} \gamma_k h_k(x) \right)^2 \right] \; \mathit{with}\, D_R = \bigcup_{k' \in S_R} D_{k'} \tag{9}$$

A problem of this type can be solved efficiently in a federated learning context with peer review by means of the algorithm reported in Algorithm 2.

Similar to the gradient boosting algorithm, the review phase makes it possible to carry out a line search useful for determining the optimal weights of the contribution of each weak learner to the prediction of the global model. However, these values are determined by generally considering only a subset of all available clients, for this reason it is possible to add a regularisation factor to the model by scaling each of the constants γ_k learned at iteration m by a multiplicative factor η which takes the name of learning rate or shrinkage factor.

4.3. Regression

To prove the FedLSBT effectiveness, we consider applying it to a regression task using the California Housing Prices dataset[5]. The objective of the task is to predict the continuous variable corresponding to the average price of a house for each district of California, starting from eight features that concern the characteristics of the houses in the district. We train the model using the Mean Squared Error (MSE) objective function. Moreover, to better simulate a real federated

[5]https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset

Algorithm 2 FedLSBT training

```
Input: clients set $S$ indexed $1...K$, local training set for each client $D_k = \{(x_i, y_i)\}_{i=1}^{N_k}$, differentiable loss function $L(y, F(x))$, number of iterations $M$, fraction of client participating in the training round $f_T$, fraction of client participating in the review round $f_R$, learning rate $\eta$.
Output: learned global model $F_M(x)$.
  procedure FedLSBT($S$, $M$, $f_T$, $f_R$)
    Initialize model with a constant value: $F_0(x) = 0$
    for $m \leftarrow 1$ to $M$ do
      Sample a subset of clients $S_T \subseteq S : |S_T| = \lfloor f_T \cdot |S| \rfloor$
      Initialise set of learned updates to the global model: $H_m \leftarrow \emptyset$
      for $k \in S_T$ do
        Send global model $F_{m-1}(x)$ to client $k$
        Update $H_m$: $H_m \leftarrow H_m \cup \text{ClientTrain}(k, F_{m-1}(x))$
      end for
      Sample a subset of clients $S_R \subseteq S : |S_R| = \lfloor f_R \cdot |S| \rfloor$
      Initialise: $C \leftarrow 0 \in \mathbb{R}^{|S_T| \times |S_T|}$, $R \leftarrow 0 \in \mathbb{R}^{|S_T|}$
      for $k' \in S_R$ do
        Send $F_{m-1}(x)$ and $H_m$ to client $k'$
        $R_{k'}, C_{k'} \leftarrow \text{ClientReview}(k', F_{m-1}(x), H_m)$
        Update $R$ and $C$: $R \leftarrow R + R_{k'}$, $C \leftarrow C + C_{k'}$
      end for
      Compute learning rates: $\gamma = \frac{1}{|S_T|}(C^T C)^{-1} R$, $\gamma \in \mathbb{R}^{|S_T|}$
      Update model: $F_m(x) = F_{m-1}(x) + \eta \sum_{i=1}^{|S_T|} \gamma_i h_i(x)$, $h_i \in H_m$
    end for
  end procedure

  procedure ClientTrain($k$, $F(x)$)
    Compute pseudo-residuals: $r_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]$, $(x_i, y_i) \in D_k$
    Fit a tree $h_k(x)$ using the training set $\{(x_i, r_i)\}_{i=1}^{N_k}, x_i \in D_k$
    Send $h_k(x)$ to the server
  end procedure

  procedure ClientReview($k$, $F(x)$, $H$)
    Compute pseudo-residuals: $r_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]$, $(x_i, y_i) \in D_k$
    for $h_j(x) \in H$ do
      Compute predicted residuals on $D_k$: $\hat{r}_j = [h_j(x_0), h_j(x_1), ..., h_j(x_{N_k})]^T$
    end for
    Initialise matrix: $P \in \mathbb{R}^{N_k \times |S_T|} \leftarrow \begin{bmatrix} \hat{r}_0 & \hat{r}_1 & \dots & \hat{r}_{|H|} \end{bmatrix}$
    Compute: $C_k \leftarrow P^T P$
    Compute: $R_k \leftarrow P^T r$
    Send $R_k$ and $C_k$ to the server
  end procedure
```

learning context, a non-IID partitioning of the data was created based on the value of the latitude and longitude attributes of the samples. We set the number of federation clients to 30, out of which we select 10 clients for each training and revision step. Meanwhile, the number of federation iteration is set to 50.

In order to evaluate the performance of the model trained with the FedLSBT algorithm compared to the GBDT model implemented in the scikit-learn[6] library and trained in a centralised manner, the error is measured on a test dataset composed of a subset of the samples of the original dataset which are not used in the training phase. The centralised gradient boosting model is trained in such a way as to make the two algorithms comparable, setting a number of estimators equal to the number of trees built by the FedLSBT algorithm and setting a number of samples used for training of each tree equal to the average of the samples present in the dataset of each client. Section 4.3 shows the results of our experiments. The centralised model performs better than the FedLSBT federated model which however still manages to achieve a reasonable performance equal to 88% of the centralised model. The difference in performance between the two models is what could be expected in a federated learning context, and is mainly due to the characteristic non-IID distribution of the clients data distribution in the federated training.

Model	r^2 score	MSE
GBDT	0.72	0.26
FedLSBT	0.63	0.34

4.4. Classification

To prove the generality of the proposed algorithm, we test the performance of FedLSBT over a classification task. An example was created in which the previous regression problem is transformed into a binary classification task by assigning each sample (x_i, y_i) a class based on the function:

$$C(x_i) = \begin{cases} +1 & \text{if } y_i \geq median(Y) \\ -1 & \text{if } y_i < median(Y) \end{cases} \tag{10}$$

The objective function used for the classification task is Binary Cross Entropy (BCE), and the rest of experiment setups – e.g., data partitioning and federated training parameters – are similar to the regression task setup.

Section 4.4 shows the results of our experiments. The classification accuracy of the centralised version of the GBDT algorithm is equal to the FedLBST model which also manages to obtain a better BCE loss value. These results highlight the goodness of the proposed approach, which is capable to achieve centralized-like performance over non-IID data distribution.

Model	Accuracy	BCE
GBDT	0.81	2.47
FedLSBT	0.80	0.50

[6]https://scikit-learn.org

5. PRFL Limitations

While PRFL showed promising performance over different tasks, it does not represent a silver bullet solution. Accordingly, we here pinpoint some of its limitations. The introduction of a review phase involving multiple rounds of communication leads to increased latency and bandwidth usage. This issue represents a particularly relevant limitation when scenarios with limited network resources are considered. Several different strategies to mitigate this issue can be identified, mainly based on the smart selection of training clients and reviewers. While the revision process introduces an additional level of communication burden, there exist few different approaches in the literature for selecting effectively the clients participating to each optimisation round that aim at minimising the resource wastes [20, 21, 22]. Similarly, as the number of clients in a federation increases, the complexity and time required for validation during the review phase may become a scalability bottleneck. Finally, resource allocation fairness represents a relevant issue in scenarios where clients have varying computational and network resources. In this context, it is fundamental to ensure that no client is unfairly burdened by the reviewing process, thus requiring the definition of ad-hoc fairness-aware reviewers selection processes.

6. Conclusions and Future Works

In this paper, we tackle the issue of FL model performance over non-IID data distribution proposing a novel peer-review based approach for the federated learning paradigm. Our system takes inspiration from the popular procedure of peer review, common in the scientific research community, to allow the server of a federation to evaluate the local updates received by the clients in the federation. The underlying idea is to identify the relevant updates as the ones that perform well on a variety of reviewing clients, while disregarding the updates that prove to be unsuccessful over a set of various reviewers. The proposed Peer-Reviewed Federated Learning is implemented on top of Flower, obtaining the first review-enabled FL Python library. Being designed to be modular and general-purpose, the obtained library (PRFlower) is flexible enough to allow for the implementation of almost any review-based FL approach. Finally, the proposed review-based FL approach is tested on regression and classification tasks via the proposal of Federated Least Squares Boosted Trees (FedLSBT) – an extension of the popular Gradient Boosted Decision Trees (GBDT) models –, requiring federation and revision to be optimised. The simple case study highlights the validity of the proposed approach, which achieves performance similar to the centralised learning setup even with non-IID data.

In the future, we plan to extend the experimental evaluation of PRFL by testing it over a broad set of tasks and hyperparameter setups, and against a set of FL training protocols. We also aim at extending the in-depth analysis of PRFL performance to domains different from the simple tabular data used this far, including computer vision [23, 24], graph processing [25, 26], natural-language processing [27, 28], and neuro-symbolic models [29, 30].

Acknowledgments

This work was partially supported by PNRR – M4C2 – Investimento 1.3, Partenariato Esteso PE00000013 – "FAIR—Future Artificial Intelligence Research" – Spoke 8 "Pervasive AI", funded by the European Commission under the NextGenerationEU programme, and by the CHIST-ERA IV project "Expectation" – CHIST-ERA-19-XAI-005 –, co-funded by EU and the Italian MUR (Ministry for University and Research).

References

[1] L. Li, Y. Fan, K. Lin, A survey on federated learning, in: 16th IEEE International Conference on Control & Automation, ICCA 2020, Singapore, October 9-11, 2020, IEEE, 2020, pp. 791–796. doi:10.1109/ICCA51439.2020.9264412.

[2] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, Y. Gao, A survey on federated learning, Knowledge-Based Systems 216 (2021) 106775. doi:10.1016/j.knosys.2021.106775.

[3] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, G. Srivastava, A survey on security and privacy of federated learning, Future Generation Computer Systems 115 (2021) 619–640. doi:10.1016/j.future.2020.10.007.

[4] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, B. He, A survey on federated learning systems: Vision, hype and reality for data privacy and protection, IEEE Transactions on Knowledge and Data Engineering 35 (2023) 3347–3366. doi:10.1109/TKDE.2021.3124599.

[5] A. Hilmkil, S. Callh, M. Barbieri, L. R. Sütfeld, E. L. Zec, O. Mogren, Scaling federated learning for fine-tuning of large language models, in: E. Métais, F. Meziane, H. Horacek, E. Kapetanios (Eds.), Natural Language Processing and Information Systems - 26th International Conference on Applications of Natural Language to Information Systems, NLDB 2021, Saarbrücken, Germany, June 23-25, 2021, Proceedings, volume 12801 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 15–23. doi:10.1007/978-3-030-80599-9_2.

[6] X. Ma, J. Zhu, Z. Lin, S. Chen, Y. Qin, A state-of-the-art survey on solving non-IID data in federated learning, Future Generation Computer Systems 135 (2022) 244–258. doi:10.1016/j.future.2022.05.003.

[7] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-IID data, CoRR abs/1806.00582 (2018). arXiv:1806.00582.

[8] Y. Zhu, C. Markos, R. Zhao, Y. Zheng, J. J. Q. Yu, FedOVA: One-vs-all training method for federated learning with non-IID data, in: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021, IEEE, 2021, pp. 1–7. doi:10.1109/IJCNN52387.2021.9533409.

[9] W. Zhang, X. Wang, P. Zhou, W. Wu, X. Zhang, Client selection for federated learning with non-IID data in mobile edge computing, IEEE Access 9 (2021) 24462–24474. doi:10.1109/ACCESS.2021.3056919.

[10] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, N. D. Lane, Flower: A friendly federated learning research framework, CoRR abs/2007.14390 (2020). arXiv:2007.14390.

[11] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, Y. Khazaeni, Federated learning

with matched averaging, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020. URL: https://openreview.net/pdf?id=BkluqlSFDS.

[12] L. Muñoz-González, K. T. Co, E. C. Lupu, Byzantine-robust federated machine learning through adaptive model averaging, CoRR abs/1909.05125 (2019). arXiv:1909.05125.

[13] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. Agüera y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Proceedings of 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*, 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf.

[14] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, A. T. Suresh, SCAFFOLD: stochastic controlled averaging for federated learning, in: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of *Proceedings of Machine Learning Research*, 2020, pp. 5132–5143.

[15] M. Safaryan, R. Islamov, X. Qian, P. Richtárik, FedNL: Making Newton-type methods applicable to federated learning, in: International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA, volume 162 of *Proceedings of Machine Learning Research*, 2022, pp. 18959–19010. URL: https://fl-icml.github.io/2021/papers/FL-ICML21_paper_28.pdf.

[16] C. He, S. Li, J. So, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, S. Avestimehr, Fedml: A research library and benchmark for federated machine learning, CoRR abs/2007.13518 (2020). arXiv:2007.13518.

[17] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, J. Passerat-Palmbach, A generic framework for privacy preserving deep learning, CoRR abs/1811.04017 (2018). arXiv:1811.04017.

[18] J. H. Friedman, Stochastic gradient boosting, Computational Statistics & Data Analysis 38 (2002) 367–378. doi:10.1016/S0167-9473(01)00065-2.

[19] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine Learning 63 (2006) 3–42. doi:10.1007/s10994-006-6226-1.

[20] F. Lai, X. Zhu, H. V. Madhyastha, M. Chowdhury, Oort: Efficient Federated Learning via Guided Participant Selection, in: 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021, USENIX Association, 2021, pp. 19–35. URL: https://www.usenix.org/conference/osdi21/presentation/lai.

[21] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, K. Chan, Adaptive Federated Learning in Resource Constrained Edge Computing Systems, IEEE Journal on Selected Areas in Communications 37 (2019) 1205–1221. doi:10.1109/JSAC.2019.2904348.

[22] Y. J. Cho, J. Wang, G. Joshi, Client selection in federated learning: Convergence analysis and power-of-choice selection strategies, CoRR abs/2010.01243 (2020). arXiv:2010.01243.

[23] C. He, A. D. Shah, Z. Tang, D. F. N. Sivashunmugam, K. Bhogaraju, M. Shimpi, L. Shen, X. Chu, M. Soltanolkotabi, S. Avestimehr, FedCV: A federated learning framework for diverse computer vision tasks, CoRR abs/2111.11066 (2021). arXiv:2111.11066.

[24] A. Agiollo, G. Ciatto, A. Omicini, *Shallow2Deep*: Restraining neural networks opacity through neural architecture search, in: D. Calvaresi, A. Najjar, M. Winikoff, K. Främ-

ling (Eds.), Explainable and Transparent AI and Multi-Agent Systems, volume 12688 of *Lecture Notes in Computer Science*, Springer, Cham, 2021, pp. 63–82. doi:10.1007/978-3-030-82017-6_5.

[25] A. Agiollo, E. Bardhi, M. Conti, R. Lazzeretti, E. Losiouk, A. Omicini, GNN4IFA: Interest flooding attack detection with graph neural networks, in: 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), IEEE Computer Society, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 615–630. doi:10.1109/EuroSP57164.2023.00043.

[26] A. Agiollo, A. Omicini, GNN2GNN: Graph neural networks to generate neural networks, in: J. Cussens, K. Zhang (Eds.), Uncertainty in Artificial Intelligence, volume 180 of *Proceedings of Machine Learning Research*, ML Research Press, 2022, pp. 32–42. URL: https://proceedings.mlr.press/v180/agiollo22a.html.

[27] M. Liu, S. Ho, M. Wang, L. Gao, Y. Jin, H. Zhang, Federated learning meets natural language processing: A survey, CoRR abs/2107.12603 (2021). arXiv:2107.12603.

[28] A. Agiollo, L. C. Siebert, P. K. Murukannaiah, A. Omicini, The quarrel of local post-hoc explainers for moral values classification in natural language processing, in: D. Calvaresi, A. Najjar, A. Omicini, R. Aydoğan, R. Carli, G. Ciatto, Y. Mualla, K. Främling (Eds.), Explainable and Transparent AI and Multi-Agent Systems, volume 14127 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 97–115. doi:10.1007/978-3-031-40878-6_6.

[29] A. Agiollo, A. Rafanelli, M. Magnini, G. Ciatto, A. Omicini, Symbolic knowledge injection meets intelligent agents: QoS metrics and experiments, Autonomous Agents and Multi-Agent Systems 37 (2023) 27:1–27:30. doi:10.1007/s10458-023-09609-6.

[30] A. Agiollo, A. Rafanelli, A. Omicini, Towards quality-of-service metrics for symbolic knowledge injection, in: A. Ferrando, V. Mascardi (Eds.), WOA 2022 – 23rd Workshop "From Objects to Agents", volume 3261 of *CEUR Workshop Proceedings*, Sun SITE Central Europe, RWTH Aachen University, 2022, pp. 30–47. URL: http://ceur-ws.org/Vol-3261/paper3.pdf.