Ensuring Fairness in Federated Learning Services: Innovative Approaches to Client Selection, Scheduling, and Rewards

Meiying Zhang*, Huan Zhao*, Sheldon Ebron*, Ruitao Xie†, Kan Yang*

*Dept. of Computer Science, University of Memphis, USA. {mzhang6, hzhao2, sebron, kan.yang}@memphis.edu
†College of Computer Science and Software Engineering, Shenzhen University, China. drtxie@gmail.com

Abstract—**Federated Learning (FL) Services enable customers (requesters) to outsource their FL tasks to the FL service provider, who will recruit a group of clients with appropriate datasets to complete the FL task. For a given FL task, how to select appropriate clients fairly becomes a challenging problem due to budget restrictions and client heterogeneity. In this paper, we propose a new client selection, scheduling, and rewarding scheme to ensure fairness through a three-stage process: 1) multi-criteria initial client pool selection, 2) data quality-oriented per-round client scheduling, and 3) performance-based rewarding. Specifically, we first define a client selection metric with multiple criteria, such as client resources, data quality, and client behaviors. Then, we formulate the initial client pool selection problem into an optimization problem that aims to maximize the overall scores of selected clients within a given budget and propose a greedy algorithm to solve it. Furthermore, we formulate the per-round client selection problem into a data quality-oriented scheduling problem that aims to improve model quality and guarantee fairness. We propose a heuristic algorithm to divide the pool into several subsets such that the federated dataset in a subset is close to an independent and identical distribution (iid) while guaranteeing each client is selected at least once in a scheduling period. In addition, we propose a performance-based payment adjustment protocol with a bonus and punishment mechanism, such that the final payment reflects the actual performance of each selected client. Our fairness analysis and experimental results show that our scheme not only can guarantee fairness but also can improve the model quality especially when data are non-iid.**

Index Terms—**Federated Learning Services, Fairness, Client Selection, Client Scheduling, Reward**

I. INTRODUCTION

Federated Learning (FL) enables multiple parties to train machine learning models collaboratively without sharing the raw training data [1], [2]. All parties train the model on their local datasets and send the local model updates to an aggregator, who will aggregate all the local model updates and send the global model to each client for the next round of training until convergence. Due to the privacy-preserving nature and the decentralized structure, the FL framework can be widely applied in many AI-driven applications where data are sensitive or legally restricted, such as smart healthcare [3], smart transportation [4], smart finance [5], and smart life [6]. However, when deploying FL in these applications,

it is challenging to find a group of clients who have the corresponding datasets and are willing to participate in an FL task.

To cope with this challenge, we envision an FL service provider that will provide FL as a service for different applications (e.g., FLaaS [7]). In an FL service system, FL task requesters (i.e., customers) send different types of FL tasks to the FL service provider with some requirements (e.g., datasets and budgets). The FL service provider will recruit an appropriate group of clients who can satisfy the task requirements. Due to the heterogeneity of clients (including computing and communication resources, dataset size, and data quality), different clients may ask for different per-round prices for an FL task. Moreover, even if a client is selected for an FL task, it does not mean the client can participate in all the FL rounds because a) only a subset of clients will be selected from the client pool to participate in an individual FL round to reduce the communication and computation costs and b) a client may be unavailable during several FL rounds due to conflicting scheduling, out-of-battery, unstable networking environments. This makes the client selection a challenging problem in the FL service. Specifically, a promising client selection solution should consider *initial client pool selection*, *client selection in each FL round*, and *client rewarding for each FL task* with the following requirements:

Model Quality: The main goal of the client selection is to maximize the quality of the final model by selecting well-performed clients within the total budget. However, the actual performance is unknown during the initial client pool selection for an FL task. Moreover, many existing client selection algorithms [8]–[12] only focus on the per-round client selection for a given FL task.

- *Client Resources*: The client resources are usually measured in terms of available CPU ratio, memory, energy, and connection status, etc. Nishio *et al.* [8] utilized resource information such as wireless channel states, CPU/GPU availability and the size of data requested from clients in client selection. Similarly, Abdulrahman *et al.* [9] considered CPU ratio, memory, energy, and data size provided by clients. These client selection algorithms aim to maximize the number of selected clients in each FL round based on computation and communication resources. However, in the FL service system, the budget

of an FL task is usually limited.

- *Data Quality*: Due to the data heterogeneity, randomly selecting a subset of clients (e.g., FedAvg [1]) or selecting clients based on their computing and communication resources may lead to biased results [10]. Although several unbiased sampling schemes are proposed based on a multinomial distribution (MD) where client probabilities correspond to their relative sample size [11], [12], the sample size does not reflect the data distribution. Local data distributions are considered in [13], [14], where a greedy algorithm is proposed to select a subset of clients by minimizing the distribution distance.

- *Client Behaviors*: Some works choose the client based on the model performance. In [15], each client trains a local model and sends the norm of its local model to the server. The server then selects a subset of users who will further send the local models. This method can reduce the communication cost of the model updates, however, it requires all the clients to participate in the training process and an extra round of norm communication from all the clients. In [16], a biased client selection method is proposed to accelerate the convergence speed by selecting a set of clients with the top-k highest loss. In FL services, the behavior of a client should be evaluated properly in both long-term behavior (i.e., accumulated reputation) and short-term behavior (e.g., model performance in each round, drop-off rate in an FL task).

Fairness: We consider fairness in FL services during client selection and rewarding as follows: 1) each client who satisfied the FL task requirement has the chance to be selected into the initial client pool for an FL task; 2) each client has a fair chance to be selected during a given FL task; and 3) the final payment or reward should consider the performance in each participated round (i.e., with penalty and bonus on top of the asked price). Many existing client selection methods may prefer a specific type of clients (e.g., with high computing/communication resources [8], [9], low non-iid data distribution [13], [14], or high model performance [15]), which makes it unfair for those clients whose resource or data quality may not be the best, especially for FL services. In [17], the authors consider fairness during the client selection in each FL round and aim to minimize the average model training and exchange time subjecting to long-term fairness guarantee and client availability. However, the fairness in [17] is limited to ensuring that each client is selected a specific number of times. In [18], a reputation-based client selection is proposed to prioritize clients with high reputations but also provide opportunities for clients with low reputations. In a recent work [19], the authors further use Lyapunov optimization to dynamically adjust client selection probabilities by jointly considering reputations, times of participation and contributions.

In this paper, we propose to ensure the fairness of FL services by designing a multi-criteria client selection, data quality-oriented client scheduling, and performance-based rewarding scheme, which consists of three stages: 1) selecting an initial client pool for each FL task by mapping requirements from the FL requester with the client capabilities (e.g., resources, data, previous behaviors, prices) under a given budget; 2) selecting a subgroup of clients in each learning round of an FL task based on the data quality, model quality and behavior in previous rounds of this FL task. Specifically, the client pool will be divided into several subsets which aim to make the federated dataset in a subset close to the iid distribution. In a scheduling period, each subset will take turns participating in one training round. Then, we compute a reputation score for each participating client according to their performance measured by model quality and behavior. Clients who are unavailable during the next scheduling period or have low reputation scores in the previous scheduling period are temporarily removed from the client pool and added back after one or a few scheduling periods; and 3) payment adjustment based on the overall performance in all participating rounds of an FL task. Table I compares the fairness and optimization goals between our scheme and existing methods.

The contributions of this paper are summarized as follows:

- We define a client selection metric based on multiple criteria, including client resources (e.g., CPU, GPU, memory, power, communication bandwidth, time availability), data quality (e.g., dataset size, data distribution) and client behaviors (e.g., number of FL tasks completed, model performance in previous FL tasks, and dropoff ratio, etc.). We further define a score function to quantify each criterion.

- We formulate the multi-criteria initial client pool selection problem into an optimization problem that aims to maximize the overall scores of selected clients in the initial client pool within a given budget from the task requester, subjecting to a minimal number of client constraints and a minimum requirement for each of the criterion scores of every client. We also propose an efficient greedy algorithm to solve this optimization problem.

- We further formulate the per-round client selection into a client scheduling problem with dynamic availability of the client pool. Then, we propose a heuristic algorithm to divide the client pool into several subsets such that each client can participate at least once while guaranteeing that the federated dataset in a subset is close to the iid distribution. We also propose a performance-based payment adjustment method to provide bonuses and penalties.

- We show that our client selection, scheduling, and rewarding algorithms can guarantee fairness. The experimental results of training CNN models on MNIST and CIFAR datasets demonstrate that our scheme can improve the model quality especially when data are non-iid.

The remainder of this paper is organized as follows. In II, we present the related work of client selection in FL. Section III describes the system model of an FL service system and the basic training process of an FL task. A client selection metric is defined in Section IV, followed by the problem

TABLE I: Comparison of Fairness and Optimization Goals

Schemes	Fairness in Per-round Client Selection				Fairness in Initial Client Pool Selection	Fairness in Rewards
	Optimization Goal	Min Selection #	Reputation	Data Quality		
[17]	Training Efficiency	✓	✗	✗	✗	✗
[18]	Model Quality via Reputation	✓	✓	✗	✗	✗
[19]	Model Quality via Reputation	✓	✓	✗	✗	✗
Our	Model Quality via Data Quality&Reputation	✓	✓	✓	✓	✓

formulation in Section V. In Section VI, we propose a solution for each formulated problem. Section VII provides the fairness analysis of our proposed client selection solution, and Section VIII provides experimental evaluation. Finally, Section IX concludes the paper.

II. RELATED WORK

In [8], the authors formulated a client selection problem to maximize the number of selected clients based on computation and communication resources when applying FL in the mobile edge computing framework. A greedy algorithm is proposed to solve the client selection problem. In [9], a multicriteria-based client selection approach is proposed to maximize the number of clients selected in each FL round. It first filters the clients based on time, then uses a linear regression model to predict whether a client is able to perform the FL task based on the CPU, memory, and energy. These methods [8], [9] require an extra round of communication of the resource information from clients to the server. Moreover, maximizing the number of clients selected in each FL round will output a better result and also require a higher cost. However, in FL services, the budget from an FL task request is usually limited. In [20], a client's successful participation probability is estimated based on the context (e.g., CPU frequency, RAM, storage, and channel information) in the current and all the previous rounds. Then, an optimization problem is formulated to select clients within a given budget for a hierarchical FL.

Randomly selecting a subset of clients (e.g., FedAvg) or selecting clients based on their computing and communication resources may lead to biased results. To guarantee unbiasedness, Li *et al.* proposed an unbiased sampling scheme based on a multinomial distribution (MD) where client probabilities correspond to their relative sample size [11]. Although MD sampling can guarantee unbiasedness in expectation, it may still cause high variance in the amount of times a client is selected in a single FL iteration. In [12], two clustered sampling algorithms (based on sample size and model similarity) are introduced to reduce variance and keep unbiasedness during the client selection in FL.

Many existing client selection methods may prefer a specific type of client (e.g., with high computing/communication resources [8], [9], low non-iid data distribution [10], [13], [21], or high model performance [15], [16]), which makes it unfair for those clients whose resource or data quality may not be the best, especially for FL services. In [17], the authors consider fairness during the client selection and aim to minimize the average model exchange time. However, the fairness is limited to ensuring that each client is selected a specific number of times. In [18], a reputation-based client selection is proposed
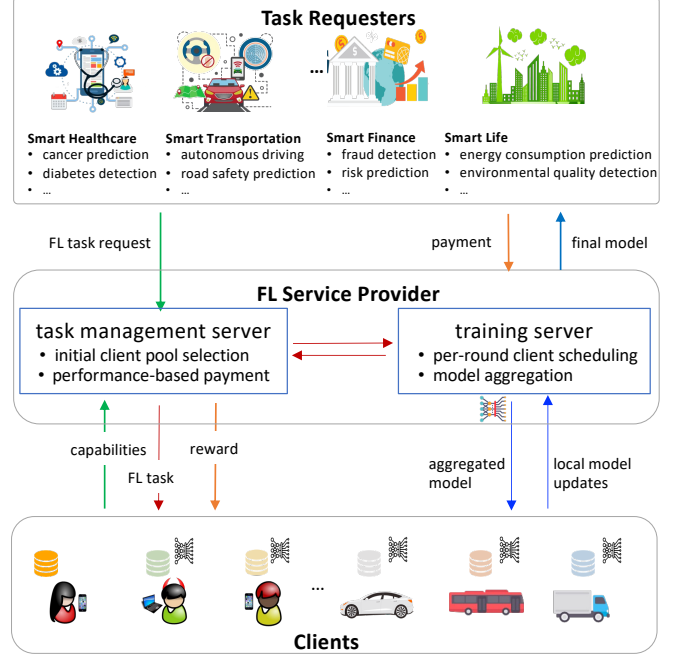

Fig. 1: System Model of FL Services

to prioritize clients with high reputations but also provide opportunities for clients with low reputations. In [19], the authors further use Lyapunov optimization to dynamically adjust client selection probabilities by considering reputations, times of participation and contributions.

III. SYSTEM MODEL

In this section, we first describe the system model of the FL service system. Then, we describe the basic training process of an FL task.

FL Service: As shown in Fig. 1, an FL service system consists of the following entities:

- *Task Requester*: Task requesters (i.e., customers) send different types of FL tasks to the FL service provider with some requirements (e.g., datasets, minimum computing/communication requirement, time, budget, etc.).
- *Clients*: Clients register with the FL service provider to participate in some FL tasks for profit. Each client will ask for a price that may vary for different types of FL tasks, computing/communication resources, available time, and datasets.
- *Service Provider*: The FL service provider will recruit an appropriate group of clients who can satisfy the task requirements. However, even if a client is selected for an FL task, it does not mean the client is able to participate

in all the FL rounds because a) only a subset of clients will be selected from the client pool to participate in an individual FL round in order to reduce the communication and computation costs and b) a client may be unavailable during several FL rounds due to conflicting scheduling, out-of-battery, unstable networking environments.

FL Training Process: A general FL training process happens between an aggregator and a set of clients S. Let $\mathcal{D}_k$ be the local dataset held by the client k ($k \in S$). The typical FL goal [1] is to learn a model collaboratively without sharing local datasets by solving

$$\min_{w} F(w) = \sum_{k \in S} p_k \cdot F_k(w), \ s.t. \ \sum_{k \in S} p_k = 1 \ (p_k \geq 0), \ (1)$$

where $F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(w; x^{(j_k)}, y^{(j_k)})$ is the local objective function for client k with $n_k = |\mathcal{D}_k|$ available samples. p_k is usually set as $p_k = n_k / \sum_{k \in S} n_k$ (e.g., FedAvg [1]). An FL training process usually contains multiple rounds, and a typical FL round consists of the following steps:

1) *client selection and model update*: a subset of clients S_t is selected, each of which retrieves the current global model w_t from the aggregator.
2) *local training*: each client k trains an updated model $w_t^{(k)}$ with the local dataset $\mathcal{D}_k$ and shares the model update $\Delta_t^{(k)} = w_t - w_t^{(k)}$ to the aggregator.
3) *model aggregation*: the aggregator computes the global model updates as $\Delta_t = \sum_{k \in S_t} p_k \Delta_t^{(k)}$ and update the global model as $w_{t+1} = w_t - \eta \Delta_t$, where η is the server learning rate.

IV. CLIENT SELECTION CRITERIA

As aforementioned, clients can be chosen based on various selection criteria including measures of computation resources (e.g., CPU, memory, storage, and battery/energy), communication resources (e.g., connection status, network bandwidth), data quality (e.g., dataset size and distribution) and reputation measures (e.g., historical model quality and behavior). To assist client selection in FL services, we first define a comprehensive client selection metric that consists of all of these factors. Specifically, we define a score for each of these criteria and represent them as a vector $s = (s_{criterion,1}, s_{criterion,2}, ..., s_{criterion,\#})$. where # is the number of criteria. Each score $s_{criterion,i}$ is scaled to (0, 1).

TABLE II: Notations

Score	Description
s_{CPU}	Available CPU ratio
s_{GPU}	Available GPU ratio
s_{MEM}	Memory size
s_{STR}	Storage size
s_{POW}	Power of the device (e.g., battery life)
s_{BDW}	Network bandwidth
s_{CON}	Connection status
$s_{DataSize}$	Data size
$s_{DataDist}$	Data distribution
s_{ModelQ}	Historical model quality
s_{Bhvr}	Behavior score

Table II summarizes the criteria used in this paper. Depending on the specific FL task, the server can choose one or more of the listed scores or add scores for additional criteria to consider in client selection. Optionally, the server can also combine two or more of the scores into one score as necessary for ease of computation or simplicity.

A. Definition of Resource Scores

The computation and communication resources can be obtained during the client registration or when submitting client preferences for the FL tasks. We can convert the computation and communication information into the scores by comparing them with the minimal requirements of an FL task. Suppose for a given FL task, the minimal requirement of the computation and communication resources are defined by the FL task requester as $(CPU_{min}, GPU_{min}, MEM_{min}, Storage_{min}, POW_{min}, BDWmin, CON_{min})$, the score of CPU for client i can be calculated as $s_{CPU,i} = CPU_{client_i}/CPU_{min}$.

Then, all the $s_{CPU,i}(i \in [1, n])$ are normalized into the range of (0, 1). Similarly, $s_{GPU}, s_{MEM}, s_{STR}, s_{POW}, s_{BDW}, s_{CON}$ can be calculated by the FL service provider. We can use a similar approach to compute the score $s_{DataSize}$. Other approaches can also be applied in calculating resource scores. For example, the connection channel status s_{CON} can also be measured using Shannon's equation as in [20].

B. Definition of Data Distribution Score $s_{DataDist}$

The data distribution score $s_{DataDist}$ characterizes how the client's data is independently and identically distributed (iid). It is defined as the complement of non-iid degree Nid. That is, $s_{DataDist} = 1 - Nid$ where Nid is a function of a histogram $\boldsymbol{h} = (h_1, h_2, ..., h_c)$ which represents distribution of a client's data over classes $1, 2, ..., c$ of a classification task. For example, $\boldsymbol{h} = (10, 20, ...)$ means the client has 10 data samples of the first class label, 20 data samples of the second class label, and so on. We define the non-iid degree as the fraction of difference between sample sizes of maximum and minimum classes over the total sample size.

$$Nid(\boldsymbol{h}) = (max(\boldsymbol{h}) - min(\boldsymbol{h}))/sum(\boldsymbol{h}) \qquad (2)$$

Alternatively, the non-iid degree can also be defined as a distance between two distributions, i.e., the client's data distribution $\boldsymbol{h}$ and the uniform distribution $\boldsymbol{u} = (\frac{1}{c}, \frac{1}{c}, ..., \frac{1}{c})$, such as L2 distance [13], Hellinger distance [14], Kullback–Leibler (KL) divergence [22], etc.. In this work, we use our definition (Equation 2) for simplicity and ease of understanding.

C. Definition of Historical Model Quality Score s_{ModelQ}

The FL service provider will maintain historical model quality evaluations for each FL task and every client who participated in the task. That is, for each client, there is a vector $\boldsymbol{q} = (q_{task_1}, q_{task_2}, ...)$, where q_{task_i} represents the quality of the model computed by this client for task i. This per-task model quality q_{task_i} is the average of per-round model

quality values q_t for all rounds in which the client successfully participated. Thus, for each client,

$$q_{task_i} = \frac{1}{|\mathbb{T}|}\Sigma_{t \in \mathbb{T}} q_t \tag{3}$$

where $\mathbb{T}$ is the set of indices of participated rounds.

For each round t, the model quality value q_t can be defined as a cosine or other similarity between the local and global models: $q_t = sim(\boldsymbol{w_l}, \boldsymbol{w_g})$ where $\boldsymbol{w_l}$ is the local model parameter vector (model update) computed by the client, $\boldsymbol{w_g}$ is the global model parameter vector obtained after the aggregation step for this round.

The historical model quality score s_{ModelQ} is defined as the average of all $q_{task_1}, q_{task_2}, \ldots$. The server can maintain model qualities for a fixed number $|\boldsymbol{q}|$ of recently completed tasks as $s_{ModelQ} = (q_{task_1} + q_{task_2} + \ldots)/|\boldsymbol{q}|$.

D. Definition of Behavior Score s_{Bhvr}

The behavior score measures how often a client drops out or fails to return a model in an FL task. In the same method as the model quality score is defined, a per-round behavior score b_t is computed for each participating client after each round of a task. The per-round behavior score b_t is a binary indicator of whether the client successfully returns its local model update to the server:

$$b_t = \begin{cases} 1 & \text{if model update is successful} \\ 0 & \text{else} \end{cases} \tag{4}$$

The per-round behavior scores are averaged over all rounds in which the client has participated to obtain a per-task behavior score:

$$b_{task_i} = \frac{1}{|\mathbb{T}|}\Sigma_{t \in \mathbb{T}} b_t. \tag{5}$$

which is maintained in a vector $\boldsymbol{b} = (b_{task_1}, b_{task_2})$. The overall behavior score s_{Bhvr} is the average of all or recent per-task behavior scores: $s_{Bhvr} = (b_{task_1} + b_{task_2} + \ldots)/|\boldsymbol{b}|$.

E. Overall Score and Cost

For convenience, we rewrite the scores in Table II $s_{CPU}, s_{GPU}, \ldots, s_{Bhvr}$ as $s_1, s_2, \ldots, s_{11}$. So, the score vector $\boldsymbol{s}$ is written as $\boldsymbol{s} = (s_1, s_2, \ldots, s_{11})$. We take a weighted or unweighted sum of all the above scores as an overall score $Score$ to denote how good a client is overall in all aspects:

$$Score = Score(\boldsymbol{s}) = \boldsymbol{w} \cdot \boldsymbol{s} = \sum_{i=1}^{11} w_i s_i \tag{6}$$

where the weight vector $\boldsymbol{w}$ can be defined by the server according to the specific requirements of the FL task.

Since clients contribute their resources and time when participating in an FL task, they also ask for a price (e.g., per-task price) for the FL task. Here we define the price as $Cost$. For example, the $Cost$ can be determined by a function of the overall score of a client:

$$Cost = Cost(Score) = aScore + b \tag{7}$$

where a, b are constants, $a > 0$. Alternatively, $Cost$ can be defined as any increasing function of the overall score $Score$, or a combination of one or more of the separate scores $s_1, s_2, \ldots$. Also, each client may give a price of its own as a constant. In general, clients with higher scores demand higher costs. The maximal total cost for the selected clients is subject to a budget defined by the FL task requester.

V. PROBLEM FORMULATION

In this section, we formulate the client selection, scheduling and rewarding problem based on the criteria defined in the previous section.

Stage 1: Initial Client Pool Selection for an FL Task

Given an FL task, let $\mathbb{K}$ denote the set of all clients willing to participate, and $\mathbb{S} \subseteq \mathbb{K}$ the set of selected clients for this task. At Stage 1, we aim to build a client pool from which a subset is selected to participate in each round. We require that the number of selected clients is at least n^*. The aim is to select high-score clients as many as possible within a limited cost budget B. To do this, we want to maximize the sum of overall scores of selected clients with a budget constraint and a number-of-clients constraint. We also enforce a minimum requirement $\boldsymbol{s_{th}} = (s_{1,th}, s_{2,th}, \ldots, s_{11,th})$ for each of the criterion scores $\boldsymbol{s_k} = (s_1, s_2, \ldots, s_{11})$ of each client k. We can formulate the stage-1 client selection problem as follows.

$$\max_{\mathbb{S} \subseteq \mathbb{K}} \quad \sum_{k \in \mathbb{S}} Score_k \tag{8a}$$

$$\text{s.t.} \quad \sum_{k \in \mathbb{S}} Cost_k \leq B, \tag{8b}$$

$$|\mathbb{S}| \geq n^*, \tag{8c}$$

$$\boldsymbol{s_k} \geq \boldsymbol{s_{th}}, \forall k \in \mathbb{S} \tag{8d}$$

Stage 2: Per-round Client Scheduling

At Stage 2, we will partition the client pool $\mathbb{S}$ into T subsets and schedule a subset to participate in each training round such that the subsets take turns to participate in rounds $1, 2, \ldots, T$. We refer to these T rounds as one scheduling period in which all subsets have participated once. Scheduling periods are repeated until the global model converges. In each scheduling period, we compute a reputation score for each participating client according to their performance measured by model quality and behavior. Clients who are unavailable during the next scheduling period or have low reputation scores in the previous scheduling period are temporarily removed from the client pool and added back after one or a few scheduling periods. More specifically, given an initial client pool $\mathbb{S}$ selected at Stage 1, each scheduling period includes the following steps:

- Step 1: Generate subsets $\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_T \subset \mathbb{S}$.
- Step 2: For $t = 1, 2, \ldots, T$, all the clients in the subset $\mathbb{S}_t$ participate in round t, update per-round model quality scores and per-round behavior scores for clients in $\mathbb{S}_t$.
- Step 3: Update reputation scores and availability information for all clients in $\mathbb{S}$.

- Step 4: Update client pool $\mathbb{S}$ according to their reputation scores or updated availability, including
  - removing clients that are unavailable in the next scheduling period;
  - removing clients that have bad reputation scores in the current period;
  - adding clients that have been suspended for a fixed number of rounds due to bad performance.

The reputation score is defined as the sum of per-task model quality score and per-task behavior score as defined in Equations 3 and 5: $s_{rep} = q_{task} + b_{task}$.

Subset Generation Problem: In the subset generation step, our goal is to let the union of all the subsets cover the pool to guarantee every client can participate in at least one and at most a certain number (x^*) of rounds. Also, we want each subset to have a size in a fixed range, and the overall data distribution as uniform as possible. These requirements can be summarized as follows: a) Each subset has total data distributed as uniform as possible; b) Subset size is in a certain range; and c) Each client participates at least 1, at most x^* rounds. We formulate subset generation as an optimization problem as follows.

$$\min_{\mathbb{S}_1, ..., \mathbb{S}_T} \quad max\{nid(\mathbb{S}_t)|t = 1, ..., T\} \quad (9a)$$

$$\text{s.t.} \quad |\mathbb{S}_t| \in [n - \delta, n + \delta], t = 1, ..., T, \quad (9b)$$

$$1 \leq \sum_{t=1}^{T} x_{kt} \leq x^*, \forall k \in \mathbb{S} \quad (9c)$$

In this formulation, we address the above requirement i) by minimizing the maximum non-iid degree among all the subsets. The non-iid degree of a subset $\mathbb{S}_t$ is denoted by $nid(\mathbb{S}_t)$ which is defined as the non-iid degree of the overall data distribution when data of clients in $\mathbb{S}_t$ are put together, that is, the Nid function (2) applied to the resultant vector from addition of all histograms of clients in the subset, n and δ are a desired number of clients and its tolerance of each subset, and

$$x_{kt} = \begin{cases} 1 & \text{if } k \in \mathbb{S}_t \\ 0 & \text{else} \end{cases} \quad (10)$$

The constraints (9b), (9c) provide relaxation on the subset size and number of times a client can be selected.

Stage 3: Performance-based Payment Adjustment

The problem formulation at Stage 1 selects the initial client pool based on the resource, data quality, and previous reputation. However, clients may perform differently during the practical training rounds. If the FL service provider makes the payment based on their asked prices, it is unfair to clients that behave well during an FL task. Therefore, at Stage 3, the FL service provider will adjust the final payment to the clients according to their performance across all participated rounds in the FL task. The performance-based payment adjustment can be formulated as

Given an overall budget of B for an FL task, an initial client pool S with an asked price ($Cost_k(k \in S)$) for the FL task from each client, the total number of periods n_p until model convergence, an actual selected number of rounds n_k, a behavior score b_k and an evaluation of the model performance q_k for each client k, the goal is to design a bonus and punishment mechanism for the payment adjustment, such that the final payment reflects the actual performance of each selected client.

VI. PROPOSED SOLUTIONS

A. A Greedy Algorithm to Select the Initial Client Pool

To simplify this problem, the constraint (8d) can be first taken care of by filtering scores based on the minimum requirements. Let $\mathbb{K}_f \subseteq \mathbb{K}$ denote the set of filtered clients. Then, the constraint (8c) can be satisfied by selecting the budget B to be greater than or equal to the sum of top n^* cost values among clients in $\mathbb{K}_f$.

$$B \geq \sum_{ranki=1}^{n^*} Cost_{ranki} \quad (11)$$

With this budget, the problem becomes a 0-1 knapsack problem as follows.

$$\max_{\mathbb{S} \subseteq \mathbb{K}_f} \quad \sum_{k \in \mathbb{S}} Score_k \quad (12a)$$

$$\text{s.t.} \quad \sum_{k \in \mathbb{S}} Cost_k \leq B \quad (12b)$$

The well-known dynamic programming algorithm [23] gives an exact solution with a time complexity of $O(nB)$ where $n = |\mathbb{K}_f|$. A more efficient greedy algorithm based on decreasing ratio of score to cost runs in $O(nlogn)$ and gives an approximation within $n^{-1/2}$ [24]. In this article, we use the greedy algorithm which works as follows: select as many clients as possible in non-increasing order of score-cost ratio. We provide an experiment in Section VIII to show the performance of the greedy algorithm.

B. Subset Generation Algorithm

To solve the optimization problem of subset generation (9), an algorithm is proposed where we select from the input pool one subset at a time, making sure that the non-iid degree of the subset is minimized (9a), the subset size satisfies (9b), and keeping track of the number of times each client is selected to satisfy (9c). The next subset is selected from the clients who have not been selected before, and the selection terminates when all the clients have been selected once or more.

Multidimensional Knapsack Problem (MKP) The selection of one subset from a given pool is formulated into a 0-1 multidimensional knapsack problem (MKP) [25]–[27] with subset size constraints. Specifically, a client is treated as an item, and its data histogram h, specifying how many samples of each label the client has, is treated as c-dimensional weight where c is the number of classes for the classification task, each dimension (class/label) corresponding to a knapsack.

The objective is to maximize the total data sample size of all selected clients, where the same capacity is set for all the knapsacks to encourage even distribution of data samples (weights) over classes (knapsacks). In particular, the MKP is formulated as follows.

$$\max \quad \sum_{k \in \mathbb{S}} |\boldsymbol{h_k}|_1 x_k \tag{13a}$$

$$\text{s.t.} \quad A\boldsymbol{x} \leq \boldsymbol{b} \tag{13b}$$

where $|\boldsymbol{h_k}|_1 = \sum_{j=1}^{c} h_{k,j}$ is the size of the data at client k and simply denoted as $|\boldsymbol{h_k}|$ hereinafter, x_k is a binary variable denoting whether client k is selected or not, $\boldsymbol{x}$ is a vector of x_ks for all clients, $\boldsymbol{b}$ is a vector of capacities of knapsacks, and A is the constraint matrix for the MKP problem which is constructed by putting together histograms of all clients, that is, $A = [h_1, h_2, ..., h_K]$ where $h_k = (h_{k,1}, h_{k,2}, ..., h_{k,c})^T$ is the histogram for client k, $K = |\mathbb{S}|$ is the number of clients, and c is the number of classes or knapsacks.

The subset size constraints are applied by adding a row of 1s and a row of -1s to A and correspondingly adding max and min sizes ($n \pm \delta$) to the capacity vector $\boldsymbol{b}$. In this way, solving this MKP problem is equivalent to finding a group of clients such that their accumulated sample sizes for respective class labels fill the knapsacks as evenly as possible, thus minimizing non-iid degree of the selected group. MKP is a well-known NP-hard problem and there are several open-source solvers available, such as PuLP, Gurobi, and IBM CPLEX. We use IBM CPLEX to solve our instances of MKP. The main process of our subset generation algorithm is to select the first subset of clients from the input client pool by solving a MKP, and the second subset from the rest of the clients by solving another MKP, and so on untill no clients are left.

Nid Improvement: As subsets are selected one by one, remaining clients become fewer, so the next subset is selected from a smaller pool. This will lead to less optimal solutions to later MKPs, which means larger non-iid degrees of subsets. To mitigate this issue, we introduce Nid improvement process in which Nid of a selected subset is improved by adding some clients, who have been selected previously but still available, to fill in knapsacks as needed. This is made possible by utilizing constraint (9c) which allows clients to be selected more than once. In particular, after selecting a subset from the current pool of remaining clients (clients who have not been selected so far), if Nid of the subset is greater than a threshold, the algorithm executes the Nid improvement process that: 1) finds which knapsacks are less filled (e.g., less than a certain percentage of the capacity); 2) finds clients for compensation, i.e., clients who are available for additional selection and who has data to fill the less-filled knapsacks; and 3) if found, adds these clients to the pool and selects again. The Nid improvement process may not always proceed to step 3) because there may not be any compensation clients found in step 2). In this case, the Nid cannot be improved any further with the current pool, so the improvement process terminates and the current subset is maintained. The execution
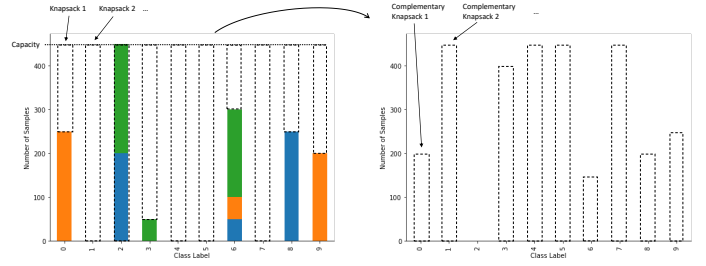

Fig. 2: Complementary Knapsacks

of the Nid improvement process makes sure that the current subset has the best Nid achievable with the remaining clients.

Complementary Knapsacks: To ensure that the MKP always has a solution, we relax the subset size constraint (9b) such that the minimum size of the subset is 1 instead of $n - \delta$, and if the solution includes less than $n - \delta$ clients even after adding compensation clients, we enforce selection of at least $n - \delta$ clients and further improve Nid by introducing complementary knapsacks method illustrated in Fig. (2).

Suppose we want to select a fixed group of mandatory clients (items) in a MKP problem. That is, as shown on the left of Fig. (2), knapsacks are not all empty, some are filled to certain degrees from the beginning. Thus, the goal becomes to find other clients to fill in available spaces of the knapsacks. The trick is to formulate another MKP by setting different capacities for new knapsacks to correspond to the available spaces of current knapsacks, that is, to complement the already filled spaces, as shown on the right of Fig. (2), and changing the input client pool to exclude the mandatory clients. Complementary knapsacks are also used when too few clients are left, in which case we select all of them and find additional clients by complementary knapsacks.

Combining the MKP formulation, Nid improvement by compensation clients and complementary knapsacks, we construct our algorithm `Generate Subsets` that successfully addresses subset generation problem (9). More detailed steps can be found in Algorithm (1) [1].

Selection of Constraint Parameters: The constraints (9b), (9c) are introduced to provide relaxation on the subset size and number of times a client can be selected, so that the problem has at least one solution. While large values of the constraint parameters δ and x^* will make the problem more possible to solve, some clients may end up being reused for too many times as a result which is not desirable for fairness guarantee. Thus, there is a trade-off between fairness and viability of the solution. Empirically, we found that δ with a value not exceeding 30% of n and x^* no more than 3 work in most cases for a 10-class classification task.

C. Performance-based Payment Adjustment Strategy

Given an overall budget of B for an FL task, an initial client pool S with an asked price ($Cost_k(k \in S)$) for the FL task from each client, the total number of periods n_p until

[1]Code is available at https://github.com/MeiyingOrAmy/Subset-Generation-for-Client-Scheduling-in-Federated-Learning

Algorithm 1 Generate Subsets

Input: Client pool $\mathbb{S}$; Histogram $\boldsymbol{h_k}$ of client $k \in \mathbb{S}$; Subset size n and tolerance δ; Max selection times per client x^*
Output: Subsets $\mathbb{S}_1, \mathbb{S}_2, ... \subseteq \mathbb{S}$
 1: Set knapsack capacities.
 2: Define data structure to track client selection status: how many times each client has been selected.
 3: $RemainingClients = AllClients$
 4: **while** $RemainingClients$ is not empty **do**
 5:    **if** $Size(RemainingClients) \geq n - \delta$ **then**
 6:       $Subset = SolveMKP(RemainingClients)$.
 7:       **if** $Nid(Subset) > NidThreshold$ **then**
 8:          Improve Nid by adding compensation clients.
 9:       **end if**
10:       **if** $Size(Subset) < n - \delta$ (selected clients are too few) **then**
11:          Obtain new $Subset$ that includes $n - \delta$ clients, by enforcing mandatory client selection.
12:          Improve Nid by complementary knapsacks.
13:       **end if**
14:    **else** $Size(RemainingClients) < n - \delta$ (too few clients left)
15:       Select all $RemainingClients$ into $Subset$.
16:       Improve Nid by complementary knapsacks.
17:    **end if**
18:    Save $Subset$ as $\mathbb{S}_i$.
19:    Update $RemainingClients$, client selection status.
20: **end while**

model convergence, an actual selected number of rounds n_k, a behavior score b_k and an evaluation of the model quality q_k for each client k, we design a bonus and punishment mechanism for the payment adjustment, such that the final payment reflects the actual performance of each selected client. The strategy consists of the following steps:

Step 1. Base Reward: According to the client scheduling algorithm at Stage 2, each client in the client pool will be selected at least once in each period for fairness. Thus, we use the number of periods n_p until model convergence as the average number of participating rounds. For client $k(k \in S)$, we can calculate the per-round cost (reward) as

$$rCost_k = \begin{cases} Cost_k/n_k & \text{if } n_k b_k \geq n_p \\ Cost_k/n_p & \text{Otherwise} \end{cases} \quad (14)$$

The base reward can be calculated as

$$Base_k = rCost_k \cdot n_k \cdot b_k, \quad k \in S.$$

On top of this base reward, we will apply penalty to bad performance and bonus to good performance. Performance is reflected by two scores: behavior score b_k and model quality score q_k. Based on the behavior score, we can obtain the 'effective' participated round as $n_k \cdot b_k$ where a client submitted the local model update successfully. Some clients may perform extra rounds beyond the average number of

rounds ($n_k \cdot b_k \geq n_p$), and some clients may drop out resulting in the effective participation less than the average ($n_k \cdot b_k < n_p$). Thus, the effective-to-average participation ratio $n_k \cdot b_k / n_p$ indicates the extra participation beyond the number of periods n_p. Similarly, among the 'effective' participated rounds, the model quality score q_k is compared with a certain threshold q_{th} (e.g., the average of all the model quality scores) to determine the relative performance of the model. That is, the ratio between the model quality and the threshold q_k/q_{th} indicates the extra model quality beyond the average.

Correspondingly, taking into account the two factors stated above: 1) extra participation beyond the number of periods n_p; and 2) model quality beyond the threshold (e.g., average) model quality, we define the performance point as

$$Ppoint_k = \frac{n_k b_k}{n_p} \cdot \frac{q_k}{q_{th}}$$

Our penalty and bonus mechanism works based on the performance point as follows.

Step 2. Penalty due to bad performance: If the performance point $Ppoint_k$ is less than 1, the per-round cost will be discounted by the amount of $Ppoint_k$. So, we have the following final reward for those clients whose performance failed to reach the average performance of all the clients (denoted as a subset S_{bad}):

$$Reward_k = Base_k \cdot Ppoint_k = Base_k \cdot \frac{n_k b_k}{n_p} \frac{q_k}{q_{th}} \quad k \in S_{bad}.$$

Step3. Bonus rewards for good performance: Due to the fixed total budget set by the FL task requester, the gap budget we can use for the bonus rewards comes from the failed participation and the punishment due to bad performance, which can be calculated as

$$B_{bonus} = B - \left(\sum_{k \in S_{bad}} Reward_k + \sum_{k \in S_{good}} Base_k \right).$$

where $S_{good} = S/S_{bad}$. This bonus budget is divided among clients in S_{good}, whose performance point $Ppoint_k$ is at least 1, based on the amount of $Ppoint_k$. So, the bonus for client k is calculated as

$$Bonus_k = B_{bonus} \cdot \frac{Ppoint_k}{\sum_{k \in S_{good}} Ppoint_k}$$

Therefore, the final reward for these clients is

$$Reward_k = Base_k + Bonus_k \quad k \in S_{good}.$$

One can easily confirm that the good performance clients receive reward beyond the base reward: $Reward_k > Base_k$ when $k \in S_{good}$, the bad performance clients receive reward under the base: $Reward_k < Base_k$ when $k \in S_{bad}$, and the better the performance, the higher the reward.

VII. ANALYSIS ON FAIRNESS GUARANTEE

We show that our proposed solutions for initial client pool selection, per-round client scheduling, and performance-based payment adjustment guarantee fairness in all parts defined previously.

Each client satisfying the minimum requirements has a chance to be included in the initial client pool. This is guaranteed by the fact that clients filtered by the minimum requirements are all considered for the optimization problem (12). Once filtered, due to the budget limit, the chance of being selected into the pool may be different between clients. In a scenario where the cost is given by individual clients, clients with a high score-cost ratio are more likely to be selected. So, those clients whose scores are low can also increase their chances by claiming a relatively lower price. In general, the chance relies on the client's score which is an overall score based on multiple criteria considered altogether. Thus, even if a client has a low score on some of the criteria, they can still improve the overall score by improving their scores on other criteria, thus increasing their chance of being selected.

Once selected into the pool, each client has a similar chance to participate in each round. Our subset generation algorithm solves the MKP problem (9) in which each client is selected into at least one subset. So, it is guaranteed that each client participates in at least one round of each scheduling period. Further, during the Nid improvement process, a small portion of the clients may be selected into additional subsets, resulting in more than one round of participation. The proportion of these clients can be kept small by controlling the values of δ and x^*. Thus, assuming an insignificant drop-out rate, most of the clients will participate in one round per scheduling period.

The final payment is based on performance in all the 'effective' participated rounds. The payment adjustment scheme in Stage 3 considers both the behavior and the model quality of each participating client in all participated rounds to assign a bonus or penalty to the final payment, such that a client who behaves well (successfully returns the model update without dropping out) and/or contributes a good quality model will receive a bonus which is obtained by deducting from payment to those whose behavior/model quality is poor. This ensures that the clients are rewarded fairly according to their performance.

VIII. EXPERIMENTAL EVALUATION

A. Experiment Settings

For Stage 1 *Initial Client Pool Selection for FL Task*, we created virtual clients by randomly assigning them scores defined in Section (IV) and compared performance of dynamic programming (DP) algorithm, random selection algorithm and the proposed greedy algorithm for selecting initial client pool from the created clients based on their scores.

For Stage 2 *Per-round Client Scheduling* We trained CNN models using MNIST and CIFAR-10 datasets distributed across 100 clients in a non-iid manner. We tested our subset generation algorithm on three different types of non-iid settings:

- Type 1 non-iid setting (one label), each client has data samples for one class label;
- Type 2 non-iid setting (two labels 9:1), each client has data samples distributed over two different labels with ratio 9:1;
- Type 3 non-iid setting (three labels 5:4:1), most clients have data samples distributed over three different labels with ratio 5:4:1, and a few clients have data samples distributed over two different labels with ratio 5:1 or 4:1.

In each type of non-iid setting, we selected 10 ± 3 clients as a subset to participate in a round, resulting in 10 to 20 subsets per scheduling period during which every client gets to participate at least one round.

The experiments were conducted in a Python environment with common ML libraries such as PyTorch, TensorFlow, Keras, etc. The basic federated learning process was implemented using a public code repository [28] on top of which we integrated our Algorithm (1) `Generate Subsets` to schedule clients for each round. We ran our experiments on a computer equippted with a 11th Gen Intel(R) Core(TM) i7 processor, 16.0 GB RAM on a Windows 11 operating system. Further details and results are presented in the subsections below.

B. Stage 1: Initial Client Pool Selection for an FL Task

In Experiment 1, to evaluate the performance of the score-cost ratio-based greedy algorithm, we generated 10 clients with random scores, calculated costs using the formula 7, with $a = 2, b = 5$, that is, $Cost = 2Score + 5$ rounded to the nearest integer for convenience, and applied the dynamic programming (DP), the greedy algorithm, and random selection under the same cost budget $B = 100$. The random selection algorithm randomly selects clients until the budget is short. In practice, a and b are chosen considering the budget, the desired number of clients to select, and the actual scores of the clients. For example, if we wish to select about 5 clients with the budget of $100, then each selected client will be allocated with about $20 on average. Since the maximum score among the 10 candidates is about 7, we can choose a and b such that $aScore_{max} + b \approx 20$.

Table III and Table IV show the input and results of Experiment 1. The dynamic programming (DP) algorithm gives the optimal solution of 6 clients with a total score of 36.85, the greedy algorithm selects 5 clients with a total score of 32.78 and an approximation ratio of 0.11 compared to the optimal solution, and the random selection results in 28.26 total score and 0.23 approximation ratio. From this example, it can be seen that, in terms of maximizing the total score, DP is undoubtedly the best, and the greedy algorithm outperforms random selection by achieving about 90% of the best total score. In the next experiment, we compare the time efficiencies of these three algorithms.

Experiment 2 measures computation time of DP, greedy, and random selection algorithms for different number of candidate clients. The scores and costs of the candidate clients are set in the same manner as in Experiment 1. The cost budget is set

TABLE III: Experiment 1 Input

Client	0	1	2	3	4	5	6	7	8	9
Score	6.92	4.89	6.8	6.08	6.9	6.08	3.74	3.36	5.26	3.39
Cost	18	14	18	17	18	17	12	11	15	11

TABLE IV: Experiment 1 Results

	Selected Clients	Total Score	Approx. Ratio
Dynamic Programming	8, 5, 4, 2, 1, 0	36.85	0
Greedy Algorithm	0, 4, 2, 5, 3	32.78	0.11
Random Selection	2, 1, 5, 7, 6, 9	28.26	0.23

(a) Our v.s. DP   (b) Our v.s. Random
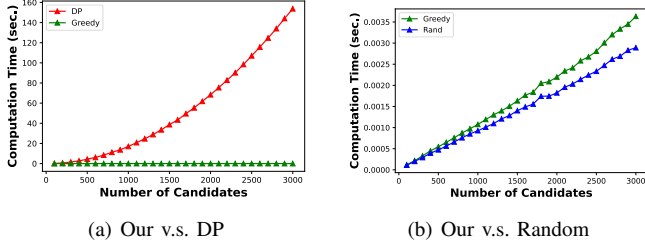
Fig. 3: Computation Time vs. # of Candidate Clients

proportional to the number of clients (candidates). Figs. 3(a) and 3(b) show how computation time changes with the number of candidates. Fig. 3(a) plots the computation time of the dynamic programming (DP) and greedy algorithm (Greedy). As expected, DP time observes quadratic increase with the number of candidates, while the greedy algorithm runs in almost the same time, compared to DP, across different sizes of input clients. Fig. 3(b) shows how the greedy algorithm looks compared with the random selection. Both algorithms are very fast but the greedy algorithm becomes slightly slower than the random as the number of clients increases, which aligns with our expectation since the greedy algorithm has a time complexity of $O(n \log n)$ and the random has that of $O(n)$ (n being the number of candidates).

C. Stage 2: Per-round Client Scheduling

We tested the subset generation algorithm on input client pools generated according to the three types of non-iid settings defined previouly. For each non-iid type, an input pool consisting of 100 clients is generated by assigning a data distribution histogram for each client.

Each of the three types of client pools is fed into Algorithm (1) to obtain a group of subsets for one scheduling period. Since the algorithm optimizes non-iid degree for each subset by re-selecting some clients, the resulting number of subsets may not be exactly $T = |\mathbb{S}|/n$, but greater than T, mostly between T and $2T$. In our experiments where the size of the pool $|\mathbb{S}| = 100$, subset size $n \pm \delta = 10 \pm 3$ and max selection times $x^* = 3$, the algorithm generated 10-20 subsets most of the time. The first step of Algorithm (1) is setting capacities of knapsacks. We set one capacity for all the knapsacks since we want them evenly filled. The capacity is set according to the client pool and desired number of rounds T. To distribute clients' data evenly across subsets, we set the capacity such that the knapsacks can accommodate data from the maximum

class (label) which is the class of which data is the most abundant across the client pool.

Fig. 4 shows examples of subsets generated by Algorithm 1 (left half) and subsets randomly selected (right half) from Types 1-3 input pools. Each bar graph indicates the total number of samples for each class label, where different colors represent data from different clients. For each pool type, the first and last subsets from Algorithm 1 are shown, the rest subsets are similar to the first. Most Algorithm 1 subsets have a close-to-uniform data distribution over labels, except for the last one which is due to the lack of remaining clients towards the end of the process. Random subsets are obtained by randomly selecting 10 clients from the pool, which have stacked histogram far away from uniform distribution.

Based on our subset generation algorithm, we scheduled clients for each round to train a FL model according to 4 steps described in Section V, one subset participating one round respectively, and then, the client pool is updated and a new set of subsets are generated for the next scheduling period. The pool update is simplified such that randomly selected 5% of the clients drop out from the pool and come back after one scheduling period. For comparison, we also trained the same model using random client selection.

For each of the 3 non-iid types, we trained a CNN model with MNIST and CIFAR datasets based on our client scheduling and random client selection. The resulting learning curves are plotted in Figs. 5 and 6. For MNIST dataset (Fig. 5), the advantage of our scheduling method is the most obvious in Type 1 pool (each client has data of only one label), the accuracy for scheduling (blue solid line) is well above that of random selection (blue dashed line), reaching an accuracy of 0.94 after 200 rounds compared to 0.78 of random. With Type 2 pool (two labels each client), the difference between scheduling and random selection reduced compared to Type 1, but still the accuracy converges faster and reaches a higher value (0.96 compared to 0.94) than random. In Type 3 pool (3 labels each client), the dashed almost caught up with the solid, but the solid shoots up a bit faster at early rounds, reaches a slightly higher accuracy (0.98 compared to 0.97), and stays more stable with less fluctuations. For CIFAR dataset (Fig. 6), with Type 1 pool, we could not reach a convergence after training for 400 rounds, but with our scheduling method, the accuracy is higher and grows faster especially towards later rounds. With Type 2 and 3 pools, our method has a better curve as expected, reaching convergence faster and achieving higher accuracy across the rounds, the difference being more significant for Type 2 than for Type 3. For both datasets, we can observe a trend that the more non-iid data individual clients have (fewer labels), the more improvement achieved by our scheduling method compared to random selection. This indicates our scheduling method works better for more extreme non-iid settings.

IX. CONCLUSION

In this paper, we focused on how to select clients fairly and improve learning performance for FL services and proposed
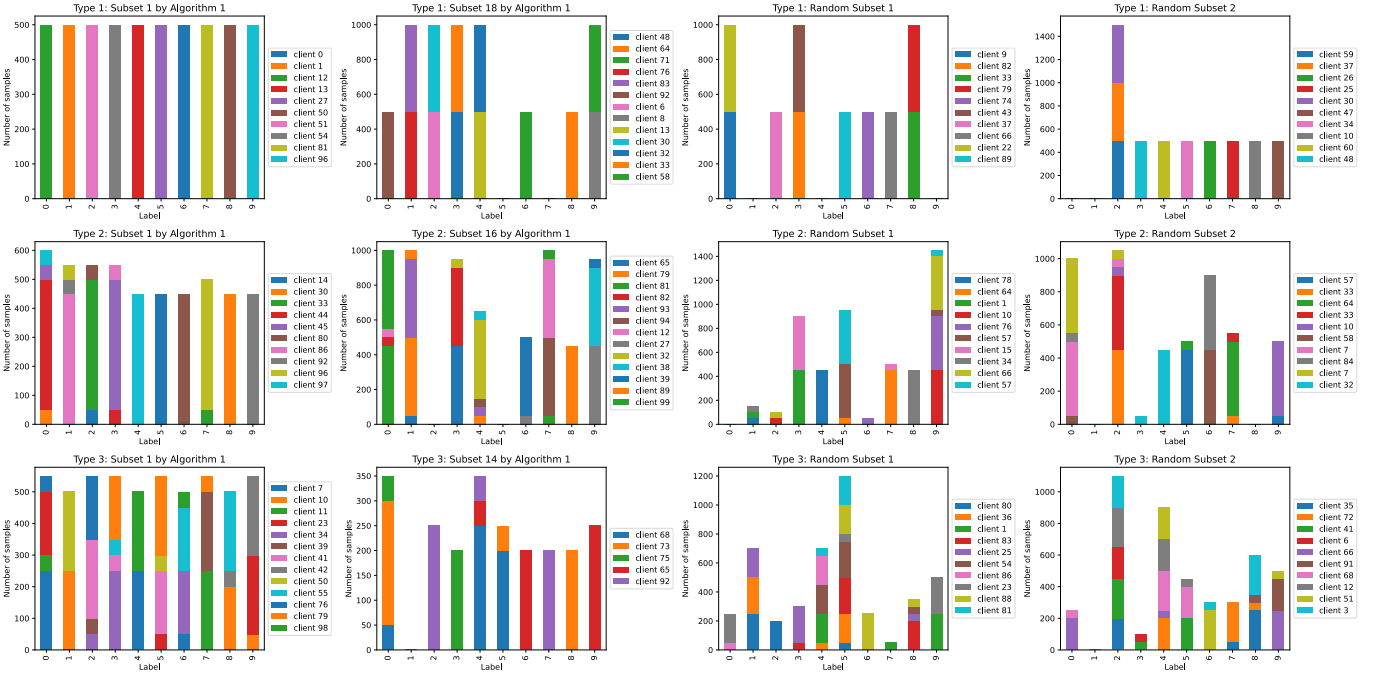
Fig. 4: Subsets Generated by Algorithm 1 and Random Subsets from Types 1-3 Pools
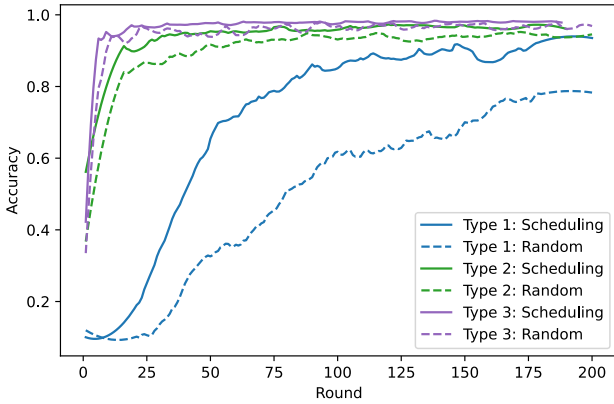

Fig. 5: MNIST Learning Curves Trained from Type 1-3 Pools
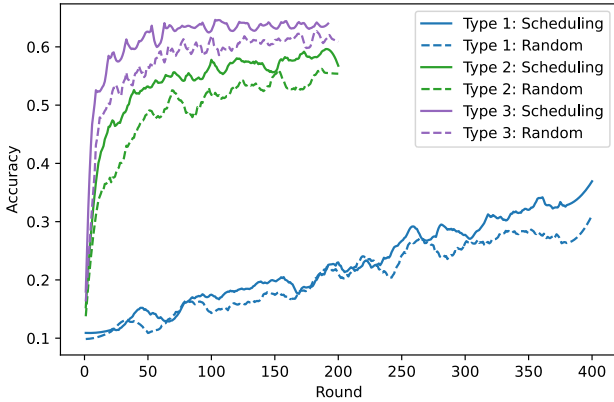

Fig. 6: CIFAR Learning Curves Trained from Type 1-3 Pools

a multi-criteria client selection and scheduling scheme with a fairness guarantee. It consists of three stages: 1) initial client pool selection, 2) per-round client scheduling, and 3) performance-based rewarding. Specifically, we first defined a client selection metric based on multiple criteria, including client resources, data quality, and client behaviors. Then, we formulated the initial client pool selection problem into an optimization problem aiming to maximize the overall scores of the initial client pool within a given budget and proposed a greedy algorithm to solve this problem. To guarantee fairness, we further formulated the per-round client scheduling problem and proposed a heuristic algorithm to generate several subsets from the client pool, while guaranteeing that the federated dataset in a subset is close to a uniform distribution and every client is selected at least once to guarantee fairness. We further proposed a rewarding mechanism to adjust the final payment according to the performance of the clients. Our fairness analysis and experimental results show that our scheme not only can guarantee fairness but also can improve the model quality especially when data are non-iid.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[2] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[3] A. M. Chekroud, R. J. Zotti, Z. Shehzad, R. Gueorguieva, M. K. Johnson, M. H. Trivedi, T. D. Cannon, J. H. Krystal, and P. R. Corlett, "Cross-trial prediction of treatment outcome in depression: a machine

learning approach," *The Lancet Psychiatry*, vol. 3, no. 3, pp. 243–250, 2016.

[4] A. Najjar, S. Kaneko, and Y. Miyanaga, "Combining satellite imagery and open data to map road safety," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[5] D. Babaev, M. Savchenko, A. Tuzhilin, and D. Umerenkov, "Et-rnn: Applying deep learning to credit loan applications," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2183–2190.

[6] M. Hino, E. Benami, and N. Brooks, "Machine learning for environmental monitoring," *Nature Sustainability*, vol. 1, no. 10, pp. 583–588, 2018.

[7] N. Kourtellis, K. Katevas, and D. Perino, "Flaas: Federated learning as a service," in *Proceedings of the 1st workshop on distributed machine learning*, 2020, pp. 7–13.

[8] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[9] S. AbdulRahman, H. Tout, A. Mourad, and C. Talhi, "Fedmccs: Multi-criteria client selection model for optimal iot federated learning," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4723–4735, 2020.

[10] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1698–1707.

[11] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[12] Y. Fraboni, R. Vidal, L. Kameni, and M. Lorenzi, "Clustered sampling: Low-variance and improved representativity for clients selection in federated learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 3407–3416.

[13] Z. Li, Y. He, H. Yu, J. Kang, X. Li, Z. Xu, and D. Niyato, "Data heterogeneity-robust federated learning via group client selection in industrial iot," *IEEE Internet of Things Journal*, 2022.

[14] J. Wolfrath, N. Sreekumar, D. Kumar, Y. Wang, and A. Chandra, "Haccs: Heterogeneity-aware clustered client selection for accelerated federated learning," in *IEEE IPDPS*, 2022.

[15] W. Chen, S. Horvath, and P. Richtarik, "Optimal client sampling for federated learning," *arXiv preprint arXiv:2010.13723*, 2020.

[16] Y. J. Cho, J. Wang, and G. Joshi, "Towards understanding biased client selection in federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 10351–10375.

[17] T. Huang, W. Lin, W. Wu, L. He, K. Li, and A. Y. Zomaya, "An efficiency-boosting client selection scheme for federated learning with fairness guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1552–1564, 2020.

[18] Z. Song, H. Sun, H. H. Yang, X. Wang, Y. Zhang, and T. Q. Quek, "Reputation-based federated learning for secure wireless networks," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1212–1226, 2021.

[19] Y. Shi, Z. Liu, Z. Shi, and H. Yu, "Fairness-aware client selection for federated learning," in *2023 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2023, pp. 324–329.

[20] Z. Qu, R. Duan, L. Chen, J. Xu, Z. Lu, and Y. Liu, "Context-aware online client selection for hierarchical federated learning," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[21] W. Zhang, X. Wang, P. Zhou, W. Wu, and X. Zhang, "Client selection for federated learning with non-iid data in mobile edge computing," *IEEE Access*, vol. 9, pp. 24462–24474, 2021.

[22] J. Lee, H. Ko, S. Seo, and S. Pack, "Data distribution-aware online client selection algorithm for federated learning in heterogeneous networks," *IEEE Transactions on Vehicular Technology*, 2022.

[23] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw-Hill Higher Education New York, 2008.

[24] J. M. Calvin and J. Y.-T. Leung, "Average-case analysis of a greedy algorithm for the 0/1 knapsack problem," *Operations Research Letters*, vol. 31, no. 3, pp. 202–210, 2003.

[25] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of heuristics*, vol. 4, pp. 63–86, 1998.

[26] M. J. Varnamkhasti, "Overview of the algorithms for solving the multidimensional knapsack problems," *Advanced Studies in Biology*, vol. 4, no. 1, pp. 37–47, 2012.

[27] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, "Knapsack problems—an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems," *Computers & Operations Research*, vol. 143, p. 105693, 2022.

[28] A. R. Jadhav, A. Portnoy, and J. Tursunboev, "Federated-learning-pytorch," https://github.com/AshwinRJ/Federated-Learning-PyTorch, 2019.