PyramidFL: A Fine-grained Client Selection Framework for Efficient Federated Learning

Chenning Li, Xiao Zeng, Mi Zhang, Zhichao Cao
Michigan State University

ABSTRACT

Federated learning (FL) is an emerging distributed machine learning (ML) paradigm with enhanced privacy, aiming to achieve a "good" ML model for as many as participants while consuming as little as wall clock time. By executing across thousands or even millions of clients, FL demonstrates heterogeneous statistical characteristics and system divergence widely across participants, making its training suffer when adopting the traditional ML paradigm. The root cause of the training efficiency degradation is the random client selection criteria. Although existing FL paradigms propose several optimization schemes for client selection, they are still coarse-grained due to their under-exploitation on the clients' data and system heterogeneity, yielding sub-optimal performance for a variety of FL applications. In this paper, we propose PyramidFL[1] to speed up the FL training while achieving a higher final model performance (i.e., time-to-accuracy). The core of PyramidFL is a fine-grained client selection, in which PyramidFL does not only focus on the divergence of those selected participants and non-selected ones for client selection but also fully exploits the data and system heterogeneity within selected clients to profile their utility more efficiently. Specifically, PyramidFL first determines the utility-based client selection from the global (i.e., server) view and then optimizes its utility profiling locally (i.e., client) for further client selection. In this way, we can prioritize the use of those clients with higher statistical and system utility consistently. In comparison with the state-of-the-art (i.e., Oort), our evaluation on the open-source FL benchmark shows that PyramidFL improves the final model accuracy by 3.68% − 7.33%, with a speedup of 2.71 × −13.66× on the wall clock time consumption.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

KEYWORDS

Machine Learning Systems, AIoT, Federated Learning, Client Selection, Data and System Heterogeneity

ACM Reference Format:
Chenning Li, Xiao Zeng, Mi Zhang, Zhichao Cao. 2022. PyramidFL: A Fine-grained Client Selection Framework for Efficient Federated Learning. In *The 28th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '22), October 24–28, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3495243.3517017

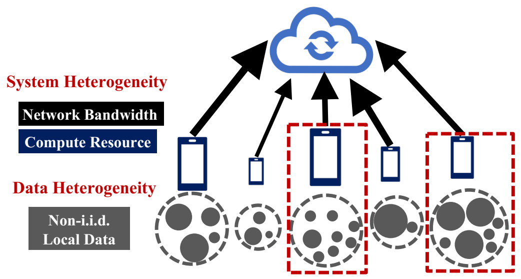


Figure 1: Illustration of the client selection problem in federated learning systems. To achieve the best performance, both data heterogeneity (non-i.i.d. data) and system heterogeneity (diverse compute resources and network bandwidths across clients) need to be jointly considered for client selection.

1 INTRODUCTION

Data privacy has become a critical concern as mobile devices and Internet-of-Things (IoT) are continuously collecting a huge amount of data from individuals on a daily basis. As a remedy to this concern, federated learning (FL) has emerged as a privacy-preserving machine learning (ML) paradigm where clients such as mobile devices and IoT distributed at different geographical locations can collaboratively train an ML model while storing their own data locally on the devices [22, 49, 54]. Such capability makes FL a core component that empowers a wide range of privacy-sensitive applications such as human activity monitoring [41, 48], home automation [52], and voice assisting [29].

At a high level, the FL process is under the coordination of a central server. In each round of federated training, the central server first distributes its current model to a crowd of selected clients; each participating client then trains the model on its own data using local stochastic gradient descent (SGD) and only sends the model update to the central server; the central server aggregates model updates from those selected clients and updates the model. These steps iterate over many training rounds until the model is converged.

FL typically involves hundreds, or even millions of clients. However, given the significant overhead of aggregating model updates from such a large number of clients, in practice, only a small fraction of clients participate in each training round [5]. Therefore, selecting *which* clients to participate in each training round is critical to both the performance and efficiency of federated training.

While existing works [4, 13, 16, 19, 24, 30, 32, 34, 35, 38, 39, 42, 48, 53] have made significant progress in many areas in FL, such as reducing the communication cost and mitigating the adverse effects of non-i.i.d. data distribution on model convergence, the majority of the existing works rely on a simple client selection strategy: in each round of federated training, a subset of participants are *randomly* selected from a large pool of available clients. Such random selection strategy, though simple, is *agnostic* to the *data and system heterogeneity* across the clients, and hence could hamper

[1]PyramidFL is available at *https://github.com/liecn/PyramidFL*

the federated training efficiency by blindly selecting clients with over-represented data or clients with slow computation speeds and limited network bandwidths [7, 8] that act as stragglers in each round of federated training.

Status Quo and their Limitations. To fill this critical gap, several client selection schemes have been proposed with different criteria under the scenario where client-side state information is available [2, 7, 8, 12, 21, 27, 40, 43, 44]. One criterion is to select clients with higher statistical utility based on various measurements such as model update importance [8, 12, 43]. Another criterion is to exploit system heterogeneity and to select clients based on their compute resources [2, 40] and communication constraints [21]. These schemes, however, are sub-optimal since data and system heterogeneity are not jointly considered. The most recent client selection scheme – Oort [27] – proposes to take both data and system heterogeneity into consideration and jointly optimizes the data and system efficiency. While Oort shows superior time-to-accuracy performance over the random selection, it is limited by its strategy in which it exploits the data and system efficiency in a *coarse-grained* manner by only taking the data and system heterogeneity *between* those selected clients and non-selected clients into consideration.

Overview of the Proposed Approach. Motivated by the limitations of existing works, in this work, we propose PyramidFL, a fine-grained client selection-based FL framework that enhances the federated training efficiency. The critical difference between PyramidFL and prior works and the fundamental idea behind PyramidFL's design is that PyramidFL takes not only the data and system heterogeneity *between* the selected and the non-selected clients but also the data and system heterogeneity *within* the selected clients into consideration. As such, PyramidFL can fully exploit both data and system efficiency in a *fine-grained* manner to improve the time-to-accuracy performance of a federated learning system.

To achieve such fine-grained client selection, PyramidFL exploits two key insights. First, a client can improve its data efficiency by training over more local data samples in one round. Second, a client that provides a less important model update for the previous model aggregation can drop some parameters for reducing the communication time without model degradation. Inspired by these two insights, PyramidFL is designed to adapt the local training processing for each participant and adopt the importance-based model update dropout to optimize the participant's data efficiency and system efficiency, respectively. Specifically, given the feedback from past training rounds, the server calculates the ranking-based configuration based on the per-client importance without leaking any data-related information. When a client receives its ranking-based configuration, it determines how many iterations to undertake to see more data samples for data efficiency and how many parameters should be dropped for system efficiency.

It is important to note that the utility of each client is not fixed but varies over training rounds: if a client has been selected, since its data will be used to train the model, its data utility will then be decreased such that the selection likelihood in the following training rounds is reduced. As such, clients who were not selected before will have a higher probability to be selected. Furthermore, PyramidFL incorporates an exploration-exploitation mechanism for client selection. Under this mechanism, PyramidFL can select clients that were not selected before to enhance the fairness of client selection further.

System Implementation and Evaluation Results. We have implemented PyramidFL using FedScale [26] and evaluated its performance on a diverse set of deep learning (DL) models and four real-world FL datasets of four important tasks, including IMU-based human activity recognition, image classification, next-word prediction, and voice command recognition. Our results show that:

- PyramidFL outperforms both random selection and Oort [27] on both time-to-accuracy and final test accuracy. Specifically, PyramidFL outperforms Oort by $2.71\times - 13.66\times$ in time-to-accuracy while achieving $3.68\% - 7.33\%$ higher final test accuracy.
- We have conducted ablation studies to validate the effectiveness of each key component incorporated in PyramidFL, demonstrating their necessity and importance to the design of PyramidFL.
- Lastly, we have conducted a series of experiments to examine the performance of PyramidFL on important system parameters. Our results show that PyramidFL is resilient to the noisy client information while achieving superior performance compared to state-of-the-art across diverse system parameters.

2 BACKGROUND AND MOTIVATION

2.1 Importance of Client Selection and its State-of-the-Art Solution

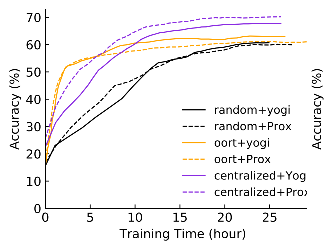
State-of-the-art client selection framework (Oort [27]) proposes a guided client selection scheme with a utility-based client selection strategy that takes data and system heterogeneity into account to select participants. Specifically, Oort associates each client C_i with a utility function designed as follows:

$$Util(C_i) = Util_{stat.}(C_i) \times Util_{sys.}(C_i) \tag{1}$$

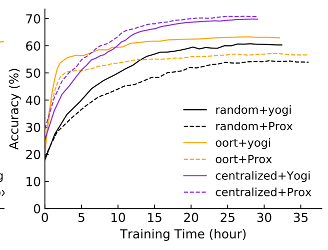
where $Util_{stat.}$ denotes client C_i's statistical utility which measures the importance of its model update, and $Util_{sys.}$ denotes its system utility which measures its speed of performing the local training and network bandwidth for communication. By selecting clients with the highest utilities, Oort can jointly maximize the data and system efficiency across clients and improve FL's time-to-accuracy performance.

To demonstrate the importance of client selection in FL, we use FedScale [26], the open-source FL benchmark to examine Oort's performance on the real-life OpenImage [25] dataset for the image classification compared to the random client selection strategy. We also include a hypothetical centralized baseline to represent the upper bound of the accuracy the trained model can achieve, in which data samples are uniformly distributed across 50 clients and trained on all the clients in each round [27]. We further consider two state-of-the-art FL optimizers (Prox [34] and YoGi [42]) and two commonly used models for mobile devices (MobileNet [45] and ShuffleNet [55]).

As shown in Figure 2, with the utility-based client selection strategy, Oort outperforms the random selection in terms of time-to-accuracy under both models with two optimizers by a large margin. Moreover, Oort can achieve a narrower gap against the centralized baseline than random selection. These results altogether demonstrate the effectiveness of selecting participants with high statistical and system utilities.



(a) OpenImage [25]+MobileNet [45].



(b) OpenImage [25]+ShuffleNet [55].

Figure 2: Importance of client selection to the time-to-accuracy performance of federated learning.

2.2 Limitations of State-of-the-Art

While Oort shows superior time-to-accuracy performance than random selection, it is not able to fully exploit both the data and system efficiency. The root cause is that Oort's statistical and system utility could only exploit the data and system efficiency in a *coarse-grained* manner: it only considers the data and system heterogeneity *between* those selected and non-selected clients. As we illustrate in the following, such a coarse-grained client selection strategy leaves a large room for improvement.

To illustrate this, we follow the settings in Oort to experiment with 120 selected participants in each round for the image classification task on OpenImage [25]. Figure 3(a) plots the ranked wall clock time consumed in one round for all 120 participants. Based on its client selection strategy, Oort first selected the top 100 clients and dropped the last 20 clients given their low system utility scores as marked by the black horizontal line.

As shown, the wall clock time consumed in one round varies significantly across those top 100 selected clients, where client#100 consumes 58.46× time than client#1. Since clients#1-#99 have to wait for client#100 to respond to the central server that completes the current round of federated training, a significant amount of time of clients#1-#99 (indicated by the grey area below the black horizontal line) that could have been utilized to improve both data and system efficiency is *wasted.*

In the following, we take a deeper look and shed light on how data and system efficiency are under-exploited in Oort. These findings serve as the critical insights for the design of PyramidFL.

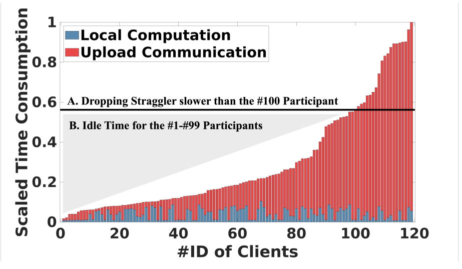
Limitation#1: Under-exploited Statistical Efficiency.

Client C_i's statistical utility in Oort is defined as:

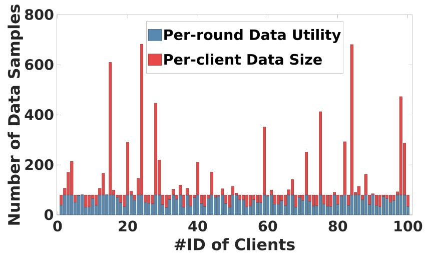
$$Util_{stat.}(C_i) = |B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2} \quad (2)$$

where B_i denotes the set of local data samples to be trained in each round, and $Loss(k)$ indicates the training loss of data sample k. As shown, the statistical utility is dependent on the number of local data samples to be trained in each round: the larger the $|B_i|$ is, the higher the statistical utility is.

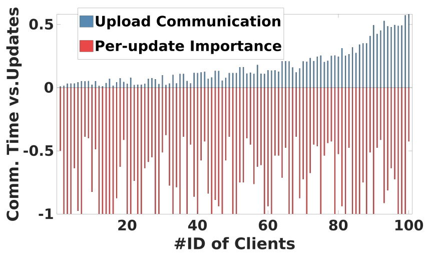
In Oort, however, $|B_i|$ is designed to be *fixed across all the clients.* To understand how data efficiency is under-exploited in Oort, Figure 3(b) plots the number of data samples seen at each selected client for the current round (denoted as per-round data utility) as well as the total number of data samples each selected client has



(a) Ranked clock time consumed by participants in one round.



(b) Under-exploited statistical efficiency.



(c) Under-exploited system efficiency.

Figure 3: Illustration of the limitations of the state-of-the-art client selection scheme.

(indicated as per-client data size). As shown, Oort's local training strategy restricts the per-round statistical utility for every client by providing a series of fixed parameters (i.e., batch size×local iteration). It should have seen more data samples for most non-straggler clients with more data than the fixed per-round statistical utility.

Limitation#2: Under-exploited System Efficiency. The system efficiency of the client C_i in Oort is defined as:

$$Util_{sys.}(C_i) = (\frac{T}{t_i})^{\mathbb{1}(T<t_i)\times\alpha} \quad (3)$$

where T is the developer-preferred duration for each round, t_i is the wall clock time consumed by client C_i for this training round, α is a developer-defined penalty factor, and $\mathbb{1}(x)$ is an indicator function that takes value one if x is true and 0 otherwise. As shown, the system utility is inversely dependent on t_i, which consists of time consumed by local computation and the communication for uploading the model update to the central server.

In Oort, each client is designed to upload the complete model update to the central server in each round. However, such a design overlooks the fact that *not all the model updates contribute to the model training equally.* Since various clients have diverse

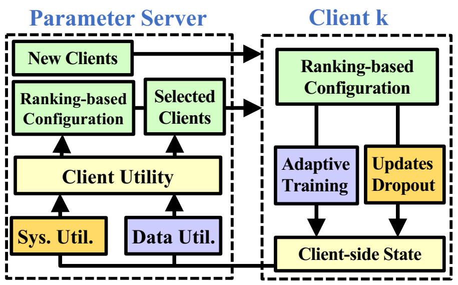


Figure 4: Overview of PyramidFL.

data sizes [27] and samples with various importance [53]. Uploading unimportant updates significantly degrades the system efficiency [30, 32, 53].

To understand how system efficiency is under-exploited in Oort, Figure 3(c) shows the inconsistency between the upload communication time and the measured model update importance (i.e., the L2-norm of the model update [3, 56]) to be communicated for aggregation. As shown, a significant amount of model updates are not essential to model training, and uploading those updates in each training round consumes time and reduces the system efficiency.

3 OVERVIEW OF PYRAMIDFL

Figure 4 illustrates the overall design of PyramidFL. At the server side, at the beginning of each training round, PyramidFL starts to select the top-ranked clients based on their utility values (§4.1) and randomly introduces new clients to participate in the federated training. PyramidFL also requires the server to compute the ranking-based configuration for participants based on client-side states from previous training rounds. The reason has two folds. First, the parameter server can collect those ranking information to consider the intra-participant divergence from the global view while avoiding data leakage (§4.2). Second, the ranking-based configuration can be used by each participant for their local training decisions. Thus each participant can update its data and system utility to adjust its likelihood to be selected for the next training round (§4.3). Specifically, each participant first retrieves their ranking-based configuration for the time consumption and then adapts its local training iterations. As such, it can fully leverage the idle time observed in Figure 3(a) to make most participants see more local data samples (Figure 3(b)). Moreover, each participant further leverages the ranking-based configuration for their model update importance to determine which parameters of its model updates should be uploaded (Figure 3(c)). This way saves valuable network bandwidth by avoiding unimportant model update uploading to fully exploit the system efficiency across selected clients. Note that we further deploy two pacers on both sides to balance the statistical and system utility and overcome the staleness of utility estimation from previous rounds (§4.4). Specifically, the pacer at the server-side can adapt the developer-defined preferred duration T of the system utility to bargain with the statistical efficiency. The pacer at the client-side can be set with several parameters to tolerate the divergence of the stale time consumption and the real one in the current round for each client.

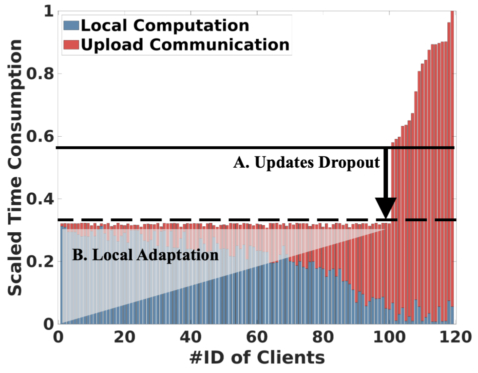
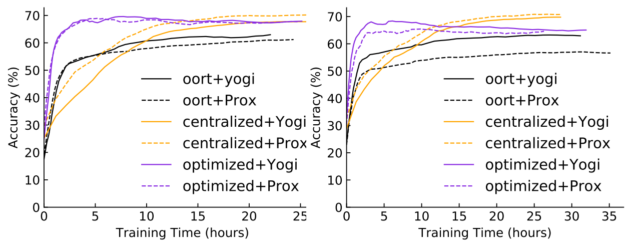


Figure 5: Illustration of PyramidFL's improvements over Oort [27] on the wall clock time consumption.



(a) OpenImage [25]+MobileNet [45]. (b) OpenImage [25]+ShuffleNet [55].

Figure 6: Preliminary validation for the optimized time-to-accuracy performance of PyramidFL, which achieves a test accuracy close to the centralized baseline.

As shown in Figure 5, with the techniques described above, each of the selected participants in PyramidFL can enhance its data utility by leveraging the idle time (i.e., a grey area) to train the local model with more data samples and enhance its system utility by reducing its computation and communication time via dropping unimportant model parameters. In this way, it can fully utilize the intra-participant divergence among those selected clients for global model training.

To verify the effectiveness of PyramidFL's fine-grained client selection, we evaluate its performance in the same preliminary setting in Figure 2. As shown in Figure 6, such enhancements in both data and system utility directly translate into improved time-to-accuracy performance and test accuracy closer to the centralized baseline. In particular, for the OpenImage+MobileNet setting, PyramidFL consumes $4.58 - 5.36$hrs to reach the final accuracy of $67.68\% - 68.04\%$ for Yogi [42] and Prox [34], respectively. In contrast, the centralized baseline achieves $69.78\% - 70.48\%$ for the final accuracy by consuming around $17.85 - 23.19$hrs for FL training. In the following section, we describe PyramidFL's critical components in detail.

4 DESIGN DETAILS

4.1 Utility Function of Client Selection

To achieve fine-grained client selection, the key is to simultaneously exploit data and system efficiency at both server and client sides.

Server Side. For each training round, the server aggregates the model updates from those selected clients to update the global model. The desired client selection strategy makes the model reach the target accuracy as soon as possible in federated learning. To achieve the goal, a general version of the optimal list of selected clients $\mathbb{C}_{opt}$ can be formulated as follows.

$$\mathbb{C}_{opt} = \underset{\mathbb{C}}{\operatorname{argmax}}\, F_A(Update(\mathbb{C})) \;/\; max(ClockTime(\mathbb{C})) \tag{4}$$

where $F_A(Update(\mathbb{C}))$ denotes the statistical utility of clients C under the aggregation function F_A (e.g., averaging by FedAvg [38]); and $max(ClockTime(\mathbb{C}))$ denotes the slowest client's wall clock time of the current training round. To prioritize those clients with higher statistical (i.e., more important updates) and system (i.e., higher system speed) utility, the server combines both to calculate the client's utility for the client selection. Then, a list of clients $\mathbb{C}$ with the highest utilities is generated for the next training round.

Equation (4) has two hints for PyramidFL's system design. 1). how important the reported updates in the utility calculation may significantly affect the required training rounds for the data efficiency. For example, the global FL model can reach the final accuracy with only half rounds with important updates from the participants in each round. 2). how the clients report the updates in the utility calculation determines the duration of the current round for the system efficiency. For example, when each client C_i only reports 90% model parameters as its updates, we can reduce $max(ClockTime(C_i)$ by up to 10% (i.e., total computation time + 90% communication time).

By noticing this, the server should rely on more fine-grained global and local information to optimize the profiling of client utilities to avoid a sub-optimal client selection in the following rounds. For example, the server calculates importance-based ranking configuration on model updates from previously selected clients. Hopefully, the lower-ranking participant should drop more parameters if selected in the following rounds since its update is less important for global model aggregation.

Client Side. Upon receiving the latest model from the central server, those two hints in Equation (4) demonstrate that the local training strategy for each participant directly determines the importance of model update and its time consumption together with its network bandwidth. In response to Equation 4, a selected client can adapt its local training to optimize its global utility at the server-side, from two aspects:

- **On System Efficiency - $\mathbb{P}$.** Parts of model parameters $\mathbb{P}$ can be removed from those unimportant participants' uploaded updates, enabling a shorter communication time between the server and the clients, thus improving its system efficiency.
- **On Data Efficiency - $\mathbb{I}$.** We can leverage the gap between computation and communication time of clients (i.e., observed in Figure 3(a)) to do more local training iterations $\mathbb{I}$ for non-straggler participants. In this way, those non-stragglers can see data samples as many as possible to increase their statistical utility (e.g., observed in Figure 3(c)) without cost for the current round's duration.

In general, we formulate the local utility optimization problem in terms of the system efficiency $\mathbb{P}$ and data efficiency $\mathbb{I}$ for each of selected participants as follows:

$$\mathbb{I}_{opt}, \mathbb{P}_{opt} = \underset{\mathbb{I},\mathbb{P}}{\operatorname{argmin}}\Big(\sum_{i=0}^{|C|} \frac{UpdateLoss(C_i, P_i)}{LocalSample(I_i)} \times max_i(T(C_i, I_i, P_i))$$
$$T(C_i, I_i, P_i) = Time(C_i) + \Delta Comp(C_i, I_i) - \Delta Comm(C_i, P_i) \tag{5}$$

where $UpdateLoss(C_i, P_i)$ indicates the client C_i's lossy model update by removing certain update parameters configured by P_i, which is determined by the server to tolerate the lossy update for the global data efficiency. And the locally seen data sample $LocalSample(I_i)$ varies as the adaptive local training iterations I_i for client C_i. Meanwhile, increasing its local training iteration can consume extra computation time $\Delta Comp(C_i, I_i)$, along with the reduced communication time $\Delta Comm(C_i, P_i)$ with the lossy updates. As a result, $max_i(T(C_i, I_i, P_i))$ indicates the duration of the current round given the selected clients $\mathbb{C}$ and corresponding $\mathbb{P}$, $\mathbb{I}$.

4.2 Global Client Selection

Utility Function. We build our utility function over Oort [27] by considering the local training optimization at client side as follows:

$$\mathbb{C}_{opt} = \underset{\mathbb{C}}{\operatorname{argmax}}\, |\mathbf{B_i^+}| \sqrt{\frac{1}{|\mathbf{B_i^+}|} \sum_{k \in \mathbf{B_i^+}} (Loss(k)^2)} \times \Big(\frac{T}{\mathbf{t_i^-}}\Big)^{\mathbb{1}(T<\mathbf{t_i^-}) \times \alpha} \tag{6}$$

$$\mathbf{B_i^+} = Batch\_size \times I_i, \qquad \mathbf{t_i^-} = t_i^{comp} + (1 - P_i) \times t_i^{comm} \tag{7}$$

where $\mathbf{B_i^+}$ denotes the enlarged seen data samples by PyramidFL's adaptive local iteration I_i while $\mathbf{t_i^-}$ indicates the impact of dropping partial updates on the wall clock time. And $\sum_{k \in \mathbf{B_i^+}} Loss(k)^2$ indicates the client C_i's training loss averaged by its more seen data samples [38]. Compared with the coarse-grained utility function of Oort in Equation (1) to 3, PyramidFL customizes the local training processing for each participant, delivering the fine-grained utility profiling for future client selection.

Importance Ranking Calculation. Although observations in Figure 3 and Equation 6 guarantee PyramidFL's superior performance over Oort, resolving the optimal $\mathbb{I}_{opt}, \mathbb{P}_{opt}$ in Equation 5 is non-trivial due to coupled parameters for data and system efficiency. To break down the optimization in Equation 5, we heuristically derive the optimal $\mathbb{I}_{opt}, \mathbb{P}_{opt}$ by a ranking-based assignment (§4.3) to optimize the utility profiling for each selected client.

The key point is how to utilize the aggregated global information at the server to determine the ranking criteria to prioritize those clients with higher true utility. For example, the importance of model updates for each participant can be computed and stored from previous training rounds. Thus we can determine $\mathbb{P}_{opt}$ for the important partially uploaded updates to be communicated when the client is selected again. Meanwhile, we leverage importance sampling [3, 23, 56] and its well-used metric the gradient norm $||Grad(C_i)||$ to determine the ranking of each client C_i among these selected ones for the current training round. By transferring the computation of each data sample's gradient norm to the whole model update of clients, we can not only measure the importance of updates accurately but also avoid the time-consuming pass over the client data to generate the gradient norm of every sample in the importance sampling [53]. We further collect the computation and communication time t_i of each selected client from the past rounds, reflecting its device type and network quality. Given the preferred time duration T in Equation (5), we can compute the estimated idle time $T - t_i$ for each client C_i and adapt its local training iterations I_i for more trained data samples. Note that the preferred time duration T varies as the FL training evolves (§4.4) and has been updated by the pacer in Figure 4 to tolerate the staleness of recordings from previous training rounds.

How PyramidFL preserves the privacy? In addition to the necessary shared information in FL (i.e., the model updates), PyramidFL also aggregates the training loss from those selected clients to the server and distributes the ranking-based configuration to each selected client C_i, without revealing the raw data in real FL deployments. And no privacy information (i.e., data distribution) about the clients can be disclosed by the shared ranking-based configuration to each client. Since the ranking criteria are determined and stored at the server.

4.3 Local Utility Optimization

Upon receiving the globally shared information from the central server, each participant uses a heuristic approach to solve the optimization problem depicted in Equation 5. First, we calculate the P_i to minimize $UpdateLoss(C_i, P_i)$ while maximizing $\Delta Comm(C_i, P_i)$. Second, according to the estimated $T - \Delta Comm(C_i, P_i)$, we determine I_i to maximize $LocalSample(I_i)$ while controlling the resulted $\Delta Comp(C_i, I_i)$ to keep that $max_i(T(C_i, I_i, P_i))$ is smaller than the developer-preferred duration T.

Ranking based Dropout: Given the importance-based ranking via the gradient norm, a selected participant can determine its communicated updates locally. For example, the ranking #1 participant should have the most parameters to upload since it can be the most important client in the next round. Thus we can save the communication time while suppressing the updated loss for aggregation in Equation (5). To achieve the parameter dropout, we leverage the well-used Dropout scheme in the ML literature [6, 47], which only requires the percentage of dropped parameters and prevents the over-fitting of the training model. Specifically, a selected client C_i computes the update parameter P_i linearly to indicate the percentage of dropped ones based on its ranking, as follows:

$$P_i = a + \frac{b-a}{|\mathbb{C}|} \times Rank(|\mathbf{B}_\mathrm{i}^+|\sqrt{\frac{1}{|\mathbf{B}_\mathrm{i}^+|}||Grad(C_i)||^2}) \tag{8}$$

where the positive a, b denote the low and up bounds of an assigned dropout ratio, corresponding to the approximate dropout ratio of the most and least important participant for this round. Note that such a ranking-based dropout can alleviate the estimation error with the stale $\mathbf{B}_\mathrm{i}^+$ and $||Grad(C_i)||$ from previous rounds.

Adaptive Local Training: Upon deriving the dropout ratio of the uploaded updates, each participant C_i can further adapt its local training iteration I_i. Thus it can further leverage the computation-communication gap in Figure 3(b) to see more local data samples. For example, each selected client can adaptively adjust its local training iteration I_i as follows:

$$I_i = (\beta \times \frac{min(T - t_i, 0)}{t_{comp.}} + 1) \times I_{fix} \tag{9}$$

where T and I_{fix} indicate the shared preferred duration for the current round and the fixed local training iteration [27]. And β is the confidence factor to tolerant the stale clock time t_i from the previous round for the current round.

Algorithm 1: PyramidFL: fine-grained client selection.

```
Input: Client set $\mathbb{C}$, participant size K, exploitation factor $\epsilon$,
       server pacer step $\Delta$, straggler penalty $\alpha$, update
       dropout bounds a,b, confidence factor $\beta$, local
       training iteration base $\mathrm{I}_{fix}$
Output: Participants $\mathbb{C}_{opt}$, Dropout $\mathbb{P}$, Adaptive Iterations $\mathbb{I}$
/* Initialize global variables */
1  $\mathbb{C}_E \leftarrow \emptyset$; $Util \leftarrow \emptyset$;   // Explored clients and total utility
2  $\mathbb{F}_{stat.} \leftarrow \emptyset$; $\mathbb{F}_{sys.} \leftarrow \emptyset$; // Feedback on stat. and sys. utility
3  $\mathbb{R} \leftarrow \emptyset$; $T \leftarrow \Delta$; // Ranking info. and preferred round duration
/* Global Client Selection at the server */
4  Function SelectionAtServer ($\mathbb{C}$,K,$\epsilon$, $T$)
5      $\mathbb{F}_{stat.}$,$\mathbb{F}_{sys.}$=GetClientFeedback()
       /* Selection #1: update client info. */
6      $\mathbb{C}_E$, $Util_{stat.}$, $t = UpdateClient(\mathbb{C}, \mathbb{F}_{stat.}, \mathbb{F}_{sys.})$
       /* Update the preferred duration T based on feedback */
7      $T$=UpdatePreferDuration($\mathbb{F}_{stat.}$, T, $\Delta$);
       /* Selection #2: update client utility. */
8      Loop client i ∈ $\mathbb{C}_E$ :
9          $Util(i)$=$|\mathrm{B}_i^+| \times Util_{stat.}^i \times (\frac{T}{t_i})^{\mathbb{1}(t_i > T) \times \alpha}$
       /* Selection #3: exploit $\epsilon$×K clients by utility. */
10     $\mathbb{C}^*$ = SelectForExploit($\mathbb{C}_E$, $Util$, $\epsilon \times K$)
       /* Selection #4: explore $(1-\epsilon)$×K new clients. */
11     $\mathbb{C}_{opt}$ = $\mathbb{C}^* \cup SelectForExplore(\mathrm{C}_E, Util_{sys.}, (1-\epsilon) \times K)$
12     $\mathbb{R}_{stat.}$=RankingClients($\mathbb{F}_{stat.}$)
13     Return $\mathbb{C}_{opt}$,$\mathbb{R}_{stat.}$,$T$
/* Local Utility Optimization for each client $C_i$ */
14 Function OptimizationAtClient ($\mathbb{C}*$,$R_{stat.}^i$,$a$,$b$,$\beta$, $T$)
15     $P_i$=$a$+$\frac{b-a}{|\mathbb{C}_{opt}|}$×$R_{stat.}^i$
16     $I_i = (\beta \times \frac{min(T - t_i, 0)}{t_{comp.}} + 1) \times I_{fix}$
17     Return $\mathbb{F}_{stat.}^i$,$\mathbb{F}_{sys.}^i$=LocalTraining($I_i$,$P_i$)
```

4.4 Put All the Pieces Together

Lastly, we summarize the complete fine-grained client selection scheme proposed by PyramidFL in Algorithm 1. Specifically, at the beginning of each training round, PyramidFL collects the statistical and system feedback from the last round (Line 5, i.e., training loss for client importance profiling, computation and communication time consumption), and updates the information for these explored clients (Line 6), including the statistical utility, consumed wall clock time. Analogous to Oort [27], PyramidFL's pacer can adapt the preferred round duration T with the pacer step Δ to balance the straggler penalty and the statistical utility (Line 7). For example, when the accumulated statistical utility in the past certain rounds decreases, the pacer allows a larger $T \leftarrow T + \Delta$ by Δ to bargain with the statistical efficiency and the system efficiency. Then PyramidFL computes the client utility as Equation 7 indicates and selects the top $\epsilon \times K$ participants sorted by its statistical and system utilities

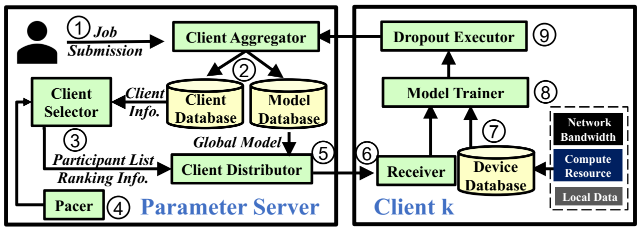


Figure 7: The system architecture of PyramidFL.

(Line 8-10). To introduce new clients for federated training, PyramidFL randomly explores $(1-\epsilon)\times K$ clients that have not been selected before (Line 11). We can also prioritize those unseen clients with faster system speed when possible (e.g., with known device models) [27] for exploration. Next, we compute the ranking-based configuration in Equation (8) for the local utility optimization in the current training round. Thus the selected clients can make their local training decisions locally. For example, each selected client C_i computes the dropout ratio for model update parameters to be communicated and adapts its local training iteration for wall clock time balance (Line 15-16). In this way, it can see more data samples during training while only uploading important updates to the central server (Line 17).

5 SYSTEM IMPLEMENTATION

We have implemented PyramidFL using FedScale [26], a benchmark and open-source evaluation platform for federated learning. Figure 7 shows PyramidFL's system architecture. Specifically, PyramidFL adopts the parameter server (PS) architecture. ①: at the PS side, the developer first submits the FL job along with the specific criteria for client selection to the *Client Aggregator*. ②: Along with the clients' feedback from the last training round, the *Client Aggregator* collects and updates the client information and global model, respectively. ③: the *Client Selector* generates a list of eligible clients meeting the developer's utility criteria (e.g., statistical and system utility) for the next training round. ④: the *Pacer* controls the balance between both utilities by adjusting the developer-preferred duration for each round. ⑤: the *Client Distributor* retrieves the participant list and information as well as the most recent global model, ⑥ which can be transmitted to the *Receiver* at the corresponding client. ⑦: Given the client's local data, compute, and communication resources, ⑧ the *Model Trainer* launches its local training process with the adaptive training iteration. ⑨: Consequently, the *Dropout Executor* determines its communicated model updates given the retrieved ranking-based configuration (i.e., client importance). The whole processing is executed iteratively and tested every few rounds until it reaches the target accuracy [5].

6 EVALUATION

In this section, we evaluate the performance of PyramidFL with the aim to answer the following questions:

- **Q1 (§6.2):** *Does PyramidFL outperform the status quo and the random client selection baseline for various FL applications?*
- **Q2 (§6.3):** *How effective is each core technique incorporated in the design of PyramidFL?*
- **Q3 (§6.4):** *How is the performance of PyramidFL affected by the system hyper-parameters?*

Table 1: Summary of tasks, datasets, and ML models.

Dataset	Type	# of Clients	# of Samples
#1 OpenImage [25]	Image	7,903	1,149,938
#2 Google Speech [50]	Audio	2,187	102,851
#3 StackOverflow [9]	Text	342,477	42,719,063
#4 HARBox [41]	IMU	121	34,115

6.1 Experimental Methodology

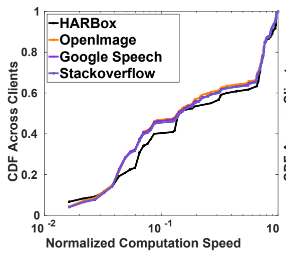
Tasks, Datasets, and ML Models. To demonstrate PyramidFL's generality across tasks, datasets, and ML models, we evaluate PyramidFL on four real-world datasets designed for FL applications at different scales. We evaluate PyramidFL across four categories of FL applications, in which each dataset relies on the collection information (e.g., HARBox [41] uses userID as the directory name) to indicate corresponding clients. Thus it follows the real non-i.i.d. data in FL scenarios and varies in data quantities, distribution, and outputs. Table 1 and 2 summarize the statistics of each dataset.

- **Human Activity Recognition**. HARBox [41] is the 9-axis IMU data collected from 121 users' smartphones for human activity recognition in a crowdsourcing manner. Such a controlled collection makes it less non-i.i.d. in all four evaluated datasets, and we can observe that it is the closest to the i.i.d data distribution in Figure 9. We further apply the resampling with a sliding time window of 2s at 50Hz to deliver a 900-dimension feature for all 34,115 data samples [41]. Considering the simplicity of the dataset and task, a lightweight customized DNN with two dense layers followed by the SoftMax layer is deployed in the FL processing.
- **Image Classification**. OpenImage [25] contains 1.1 million images from around 8,000 clients. And we train MobileNet [45] and ShuffleNet [55] models for the image classification.
- **Speech Recognition**. Google speech dataset [50] covers 100,000 audio commands from more than 2,000 clients. And ResNet-34 [14] is trained for a 20-class speech recognition task.
- **Natural Language Processing**. The large-scale StackOverflow [9] collects the posts, votes, tags, and badges on StackOverflow for the next-word predictions. Then, we train the lightweight Albert model [28] for 42 million data samples from 342,477 clients.

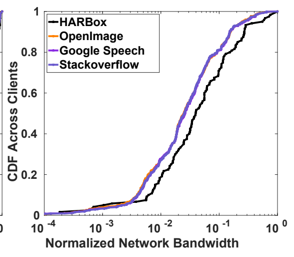
System Heterogeneity. For the local computation resource emulation, we acquire the heterogeneous runtimes of different deep learning models across hundreds of device types from the AI benchmark [18] and assign them to the clients in these datasets. Figure 8(a) illustrates the distribution of the computation efficiency across clients in our evaluated dataset. We emulate the communication time consumption for clients with various network throughput/connectivity from the network measurements on mobile devices [17]. As shown in Figure 8(b), the best communication channel can be 10,000× faster than the worst one, representing the severe impact of straggler in our evaluated FL scenarios.

Table 2: Summary of PyramidFL's improvements on time to accuracy over Oort [27]. We tease apart the overall improvement with statistical (i.e., final accuracy or perplexity) and system ones (i.e., wall clock time to reach the target final accuracy).

Federated Applications	Dataset	Model	Oort [27]+Yogi [42]		PyramidFL +Yogi		Oort [27]+Prox [34]		PyramidFL +Prox	
			Acc./Perp.	Time	ΔMetric	Speedup	Acc./Perp.	Time	ΔMetric	Speedup
Image Classification	#1 [25]	MobileNet[45]	63.16%	12.41h	**4.90%↑**	**9.13×**	60.94%	18.25h	**6.73%↑**	**13.66×**
		ShuffleNet[55]	62.39%	9.16h	**3.77%↑**	**8.78×**	56.63%	12.58h	**7.22%↑**	**6.76×**
Speech Recog.	#2 [50]	ResNet-34[14]	55.51%	22.57h	**5.55%↑**	**2.71×**	59.52%	28.19h	**3.82%↑**	**2.74×**
Word Predict.	#3 [9]	Albert [28]	43.74	46.58h	**3.68↓**	**5.79×**	38.45	78.33h	**5.49↓**	**9.31×**
Activity Recog.	#4 [41]	Customized	59.54%	12.86h	**5.23%↑**	**3.03×**	53.87%	11.87h	**7.33%↑**	**3.97×**

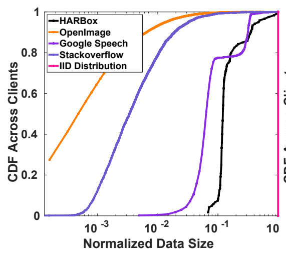


(a) Diverse Computation Efficiency

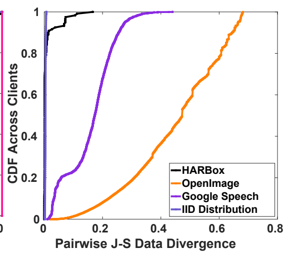


(b) Diverse Communicate Efficiency.

Figure 8: Heterogeneous system utility across clients.



(a) Heterogeneous Data Size.



(b) Categorical Data Divergence.

Figure 9: Non-i.i.d data across clients in the real-world dataset.

Data Heterogeneity. Figure 9 further profiles the non-i.i.d. data across clients. Specifically, Figure 9(a) and 9(b) measure the per-client quantity of samples [27] and the deviation of per-client categorical distributions from the average, using the popular Jensen-Shannon divergence [36]. Based on the divergence from the i.i.d. data distribution, we divide all four datasets with various kinds of non-i.i.d (e.g., medium non-i.i.d. StackOverflow) and evaluate its impact on PyramidFL's performance (§6.2).

Baselines and FL Optimizers. We compare PyramidFL with two baselines: 1) Random Client Selection and 2) Oort [27]. We also apply two state-of-the-art FL optimizers, namely Yogi [42] and Prox [34], to evaluate PyramidFL's generality across FL optimizers.

Evaluation Metrics. We use two metrics to evaluate the performance of PyramidFL and the baselines.

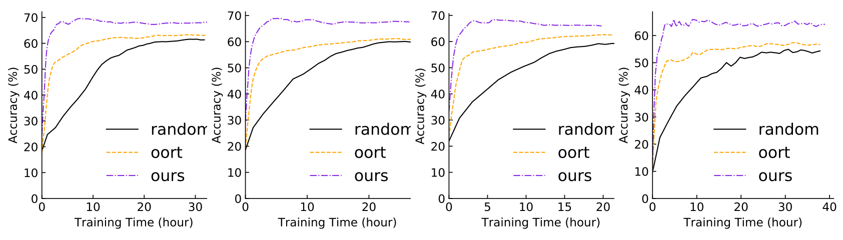
- **Time-to-Accuracy.** Time-to-Accuracy is defined as the wall clock time for training an ML model to reach a target accuracy. For simplicity, we set the target accuracy to be the achievable accuracy by all used strategies, which turns out to be the one for the random client selection. Otherwise, some may never reach that target. Analogous to Oort [27] on the FL benchmark [26], we report the simulated clock time of clients in evaluations.
- **Test Accuracy.** The test accuracy is defined as the accuracy on the test datasets obtained by the model trained through federated learning. We adopt the perplexity for the next-word prediction task, which is better with a lower value.

Parameter Settings. Unless stated otherwise, our evaluation across experiments is conducted with the settings described as follows. The mini-batch size of each training round is 16 for all tasks, with a fixed local training iteration base $I_{fix} = 5$. To alleviate the impact of stragglers, we collect updates from the first K completed participants out of $1.3K$ participants in each round. $K = 50$ participants are selected by default for each training round except the human activity recognition task with $K = 20$. The initial learning rate is $4e-5$ for Albert [28] and the customized DNN and 0.04 for other models. For the configuration selection at the server-side, PyramidFL follows Oort [27] and adopts the initial exploration factor of 0.9, which is decreased by a factor of 0.98 after each round when it is larger than 0.2. The pacer step Δ of the preferred duration is 20 rounds, and the default straggler penalty α is 2. At the client-side, participants in PyramidFL determine their partial updates to be aggregated based on the global ranking information, spanning from the dropout ratio $a = 0.1$ to $b = 0.6$ in a linear fashion. Besides, the confidence factor β is determined from 0.5-1 for all four datasets, depending on their variances across clients on the dataset size, device types, and network bandwidth.

6.2 End-to-End Performance

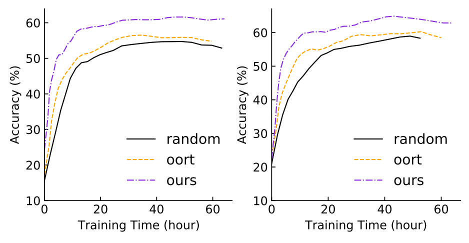
We begin by comparing the end-to-end performance of PyramidFL with both baselines on all the four datasets, shown in Table 2.

PyramidFL speeds up the wall clock time to reach the final target accuracy. Given the heterogeneous texts collected from clients in the real-world StackOverflow dataset, PyramidFL reaches the final target accuracy 5.79× and 9.31× faster than Oort in terms of wall clock time for Yogi and Prox, respectively. By providing the most diverse data size and category (See Figure 9(a)), the largest speedup is 13.66× on the OpenImage with Prox. The reason is that the high non-i.i.d. image data leaves a great potential to prioritize clients with more data samples and faster system speed. In contrast, the low non-i.i.d. HARBox, with both Yogi and Prox, delivers the slightest speedup of only 3.03× and 3.97×. The rationale is that



(a) MobileNet+Yogi. (b) MobileNet+Prox. (c) ShuffleNet+Yogi. (d) ShuffleNet+Prox.

Figure 10: Time-to-Accuracy for image classification on OpenImage dataset.



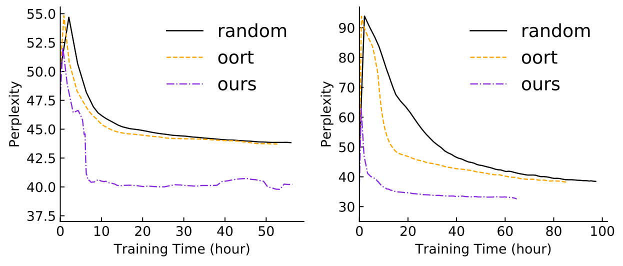
(a) ResNet-34+Yogi. (b) ResNet-34+Prox.

Figure 11: Google speech recognition.

the average samples for each client are small and approximate for the controlled HARBox collection, resulting in a limited statistical efficiency for prioritizing clients for model training speedup. Figure 10 also shows the robust speedup of PyramidFL, which is not impacted by various models and optimizers.

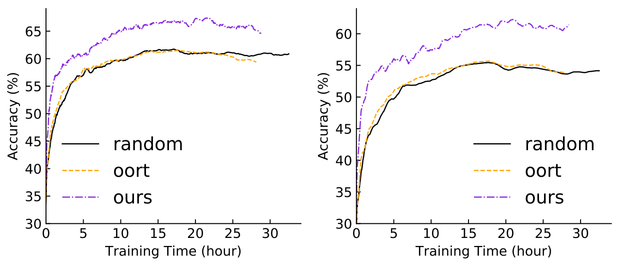
These speedups with the wall clock time reduction stem from fully exploiting the statistical and system efficiency (Table 2). PyramidFL can reach the final target accuracy with fewer training rounds than Oort and the random one, in which it further consumes less communication time for the update aggregation. Regarding various non-i.i.d. datasets, we also notice that Oort achieves the largest speedup over the random client selection on the high non-i.i.d. OpenImage [25] in Figure 10 while both are comparable on the low non-i.i.d. HARBox [41] in Figure 13. We argue that Oort's coarse-grained client selection suffers from low non-i.i.d. data by ignoring the divergence among those non-straggler selected participants while PyramidFL exploits those non-stragglers with the local utility optimization.

PyramidFL improves the final accuracy on model testing. In comparison with Oort, Table 2 shows PyramidFL achieves 4.90% and 6.73% higher final accuracy on the high non-i.i.d. OpenImage dataset with the MobileNet [45]. Considering the final accuracy can be enhanced by exploiting high statistical utility clients, real-world images often exhibit more significant heterogeneity in data characteristics than the other three types of data (i.e., audio, text, and IMU). Oort further points out that the quality of global aggregation determines the model accuracy. With only Oort's coarse-grained client selection or the random one in each round, clients with poor statistical utility can dilute aggregation quality. Therefore, the global model converges to sub-optimal performance. Instead, PyramidFL makes the models training and global aggregation with participants with more accurate utility profiling, delivering a better final accuracy for all datasets, models, and optimizers. We also notice that Oort achieves marginal accuracy improvements on the low non-i.i.d. datasets, such as the HARBox [41] and Google Speech [50], shown in Figure 11 and 13. In contrast, PyramidFL achieves higher accuracy from 3.82% to 7.33% than Oort for both datasets in Table 2. We argue that the local utility optimization enables PyramidFL to see more data samples for those selected participants with a larger local dataset.



(a) Albert+Yogi+StackOverflow. (b) Albert+Prox+StackOverflow.

Figure 12: Next-word prediction performs better with a lower perplexity in the language modeling (LM) task.



(a) Customized DNN+Yogi+HARBox. (b) Customized DNN+Prox+HARBox.

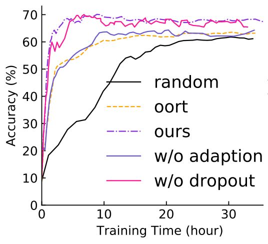
Figure 13: Human activity recognition via IMU sensors.

Table 3: PyramidFL improves time to accuracy by relying on ranking-based adaptive local training and update dropout to exploit the statistical and system efficiency, respectively.

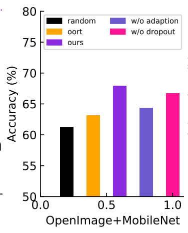
Dataset+Model	w/o Adaption		w/o Dropout	
with Yogi [42]	ΔMetric	Speedup	ΔMetric	Speedup
#1+MobileNet	**1.18%↑**	**1.52×**	**3.54%↑**	**2.95×**
#2+ResNet-34	**0.98%↑**	**1.17×**	**5.75%↑**	**2.05×**
#4+Customized	**1.02%↑**	**1.12×**	**4.95%↑**	**2.61×**

6.3 Component-wise Analysis

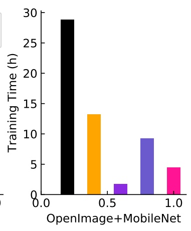
Next, we implement two breakdown versions of PyramidFL to evaluate and understand the effectiveness of each of the key components incorporated in PyramidFL.



(a) Time to accuracy.

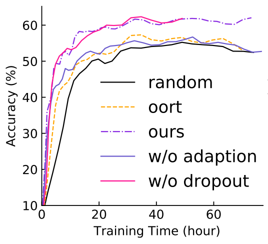


(b) Final accuracy.

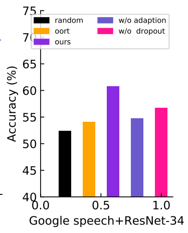


(c) Wall clock time.

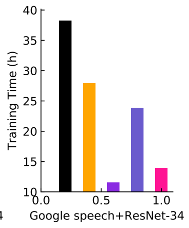
Figure 14: PyramidFL's ablation study on the OpenImage [25] for the image classification.



(a) Time to accuracy.

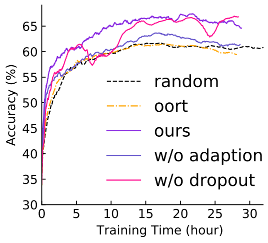


(b) Final accuracy.

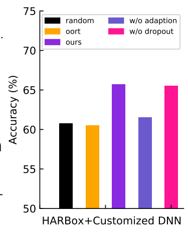


(c) Wall clock time.

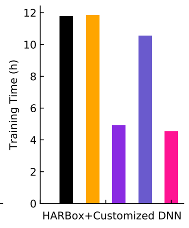
Figure 15: PyramidFL's ablation study on the Google speech [50] for the speech recognition.



(a) Time to accuracy.



(b) Final accuracy.



(c) Wall clock time.

Figure 16: PyramidFL's ablation study on the HARBox [41] for the human activity recognition.

- **PyramidFL w/o adaption.** We disable the adaptive local training, in which the local model is trained with a fixed local training iteration and cannot fully exploit the divergence of non-straggler participants. As such, it wastes a bunch of idle local computation resources, and the training can be restrained among low-utility but high-speed clients.
- **PyramidFL w/o dropout.** We remove our benefits from system efficiency by dropping partial parameters of unimportant model updates, so PyramidFL wastes too much valuable uploading network bandwidth to communicate updates, especially for those tasks with a heavy model (i.e., albert [28] for StackOverflow). We take YoGi for analysis because it outperforms Prox most of the time.

PyramidFL guarantees the finer-grained client selection via the adaptive local training for each participant. Without the local training adaption for each participant, Table 3 shows that the improvements on final accuracy and speedup decrease significantly for all three testing settings, with only around 1% and 1.1 $\sim$ 1.5×, respectively. We also plot the final accuracy distribution with the random client selection and Oort from Figure 14(c) to 16(c). Although PyramidFL w/o adaption (the dark blue) still outperforms both baselines (the black and orange), it is far below the complete PyramidFL (the purple) on both metrics. We further plot the time-to-accuracy in Figure 14(a) to 16(a). And it can be observed that PyramidFL achieves a similar trend with Oort, with a bit of improvement on final accuracy (i.e., low non-i.i.d. HARBox+Customized DNN) and wall clock time consumption (i.e., medium non-i.i.d. Google speech+ResNet-34).

Remark. The adaptive local training mainly contributes to the improvements of PyramidFL. By fully utilizing the idle time wasted by Oort, PyramidFL guarantees the optimal client selection by exploiting per-client statistical utility in each round, which is effective for non-i.i.d. FL datasets (Figure 3(a) to 6(a)).

PyramidFL optimizes the uploading communication while stabling the testing performance via the update dropout. Although PyramidFL w/o dropout achieves clear improvement on both metrics, it also demonstrates a varied testing performance as the FL training evolves. On the one hand, Table 3 shows the improvements on both metrics can reach 4 $\sim$ 6% and 2 $\sim$ 3× for PyramidFL w/o dropout. On the other hand, Figure 14(a) to 16(a) present shows the accuracy fluctuates severely as the training evolves. Taking Figure 16(a) on HARBox [41] dataset as an example, (i) At the beginning of training, both PyramidFL and (PyramidFL w/o dropout) improve the model accuracy quickly because they adaptive the local training for each client to fully exploit the statistical utility. (ii) As training evolves, the time-to-accuracy of PyramidFL w/o dropout fluctuates between the complete PyramidFL (the purple) and Oort (the yellow). For example, it decreases to the accuracy of Oort during 8 $\sim$ 12 hrs of training and increases to the one of PyramidFL after training 15 $\sim$ 22 hrs. (iii) Such a variant time-to-accuracy can be attributed to that some data samples may have already been overrepresented by the full exploitation of per-participant statistical utility over past rounds. And it can induce over-fitting training, resulting in poor performance on the testing dataset.

Remark. Although PyramidFL w/o dropout achieves greater improvements, it is not only below the performance of PyramidFL but also demonstrates an unstable performance over time due to the overrepresented unimportant model updates. By enabling dropout, we can save the valuable uploading bandwidth for update communication as well as alleviate the over-fitting of the trained model (Figure 3(c)))

By jointly optimizing the local training and update aggregation, PyramidFL can guarantee the full exploitation of statistical and system utilities for each client and stabilize the testing performance of the trained model via dropout.

6.4 Robustness and Sensitivity Analysis

We analyze PyramidFL's robustness and sensitivity to interference and parameter settings by answering the following questions: 1)

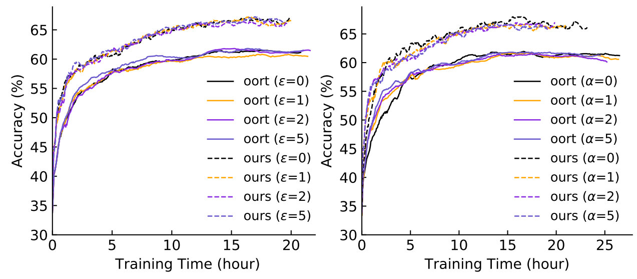

(a) Impact of noisy clients. (b) Impact of penalty factors.

Figure 17: PyramidFL performs consistently under the corrupted client utility and varying penalty factors on the straggler, comparable with Oort [27]

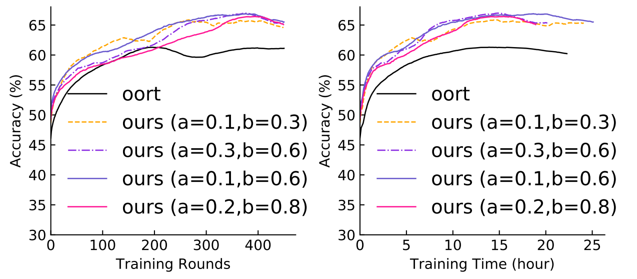

(a) Round-to-Accuracy. (b) Time-to-Accuracy.

Figure 18: PyramidFL varies performance across the dropout of update parameters.

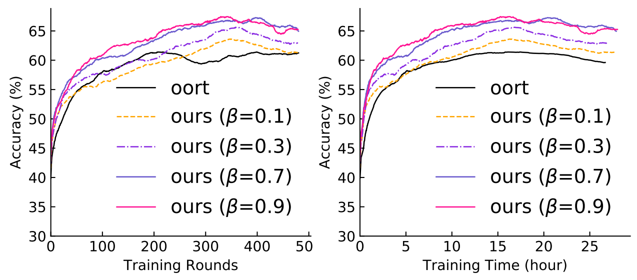

(a) Round-to-Accuracy. (b) Time-to-Accuracy.

Figure 19: PyramidFL varies performance across the local training adaptions.

Is PyramidFL resilient to the noisy utility and varied penalty for stragglers in the client selection? 2) How does PyramidFL perform as we vary the parameter settings of the adaption and dropout schemes? We take HARBox [41]+Yogi [42] for analysis, which is used for the task of IMU-based human activity recognition.

Impact of noisy utility. We first examine the impact of the noisy utility of clients, in which Gaussian noise following $(0, \sigma^2)$ are added to the collected statistical utility of participants proportionally. Such noisy utility affects the client selection for the following rounds. Similar to differential FL [11] and Oort [27], we define $\sigma = \epsilon \times Mean(utility)$, where $Mean(utility)$ is the average utility across participants for current round. As such, a larger ϵ implies a larger variance in noise. As shown in Figure 17(a), PyramidFL performs consistently under different noise levels, even under the strong one (e.g., $\epsilon = 5$ [1]). In contrast, although Oort also keeps the performance consistent with its robustness analysis, its time-to-accuracy plot shows more variance than PyramidFL (i.e., training during $2 \sim 10$ hrs).

Impact of penalty factor α. We next examine the impact of the penalty factor α, which penalizes the utility of stragglers in client selection and adaptively prioritizes clients with higher system utility. A larger α (i.e., overemphasizing system efficiency) drives to relax the system constraint T more frequently to admit clients with higher statistical efficiency (Equation (6)). As shown in Figure 17(b), PyramidFL consistently outperforms Oort across all α given its finer-grained utility computation and update for client selection.

Impact of ranking-based dropout. Equation (8) indicates that PyramidFL uses the update dropout bounds $[a, b]$ to determine the update parameter to be communicated for each participant, based on their global ranking information. Figure 18 shows that the PyramidFL's time-to-accuracy varies with different dropout bounds, while it outperforms Oort consistently. Given the global ranking information, PyramidFL orchestrates its components to assign the drop ratio between $[a, b]$ for each selected participant to navigate the best performance. For example, a larger dropout ratio saves more uploading bandwidth for communication at the expense of update loss, which should be assigned to those low-rank participants, shown in Equation (8).

Impact of ranking-based adaption. PyramidFL uses the confidence factor β to measure the divergence between the stale time consumption in the last round and the real consumption for a current round for each participant. As shown in Equation (9), PyramidFL aims to fully utilize the idle local computation resources of non-straggler participants for the update aggregation. Figure 19 shows PyramidFL's time-to-accuracy and round-to-accuracy, under various confidence factors β. First, PyramidFL suffers from a smaller β, which leverages a smaller amount of the resource with little computation time increment, making it less possible to become the slowest participants for the current round. Second, PyramidFL outperforms Oort consistently even under $\beta = 0.1$, which is almost equivalent to PyramidFL w/o adaption. Third, it performs consistently with our ablation study in Table 3. Finally, the adaptive local training contributes to PyramidFL on both metrics by fully exploiting the statistical utility of clients in a fine-grained manner.

Remark. Hyper-parameter selection is essential to PyramidFL's performance, which can be determined by the FL task types, the number of selected participants, etc. For example, the low non-i.i.d. HARBox for human activity recognition has only 5 out of 120 clients for each training round and demonstrates certain variance to the dropout ratio bounds, shown in Figure 18(b). Given its stable network quality in controlled application scenarios [41, 48], the confidence factor beta can be set as 0.9 to maximize the benefits of PyramidFL's adaptive epoch scheme. Therefore, we recommend a conservative parameter setting (e.g., a=0.1, b=0.6, β=0.7) for most FL scenarios and adapt them for small-scaled datasets accordingly.

7 RELATED WORK

Data Heterogeneity in FL. Federated learning (FL) is an emerging privacy-preserving distributed machine learning paradigm where clients (e.g., mobile devices) collaboratively train a model under the orchestration of a central server without sharing their local data [22, 49]. Given that data stored at mobile devices exhibit significant non-i.i.d. data distribution. Such data heterogeneity distinguishes FL from the existing data center-based distributed machine learning paradigm [22, 33]. To overcome the adverse effects of data heterogeneity, one primary theme of the existing works in FL focuses on designing FL optimizers at either server side or client side, such as Prox [34] and Yogi [42]. The data heterogeneity in FL does not only present challenges to the design of optimizers but also raises questions about the utility of such a global solution to individual users. Therefore, another important theme of FL is to train a personalized model for each client instead of a global model shared across all the clients. Such a personalized model can be obtained by either fine-tuning the global model using each client's local data [32, 37] or using multi-task learning [46, 52]. In our work, although PyramidFL focuses on optimizing the time-to-accuracy performance for the global model training, the techniques involved can be naturally combined with fine-tuning or multi-task learning to obtain a personalized model.

System Heterogeneity in FL. While substantial efforts have been made to address issues brought by data heterogeneity in FL, the system heterogeneity in FL has only attracted more attention recently given the diversity of devices that are potential participants of the federated training. For example, both FjORD [15] and HeteroFL [10] enable the training of heterogeneous local models with varying capacities, which differs from conventional federated learning framework where all the clients have to share the same model architecture. Given that, clients with different system resources can participate in the same federated training process. Our work also considers system heterogeneity; however, we incorporate such information for the purpose of client selection.

System Performance Optimization in FL. Since clients who participate in FL (especially in cross-device FL) are typically resource-constrained mobile and IoT devices in charging and connected to wireless networks during the federated training process, the performance bottleneck from the system perspective is mainly the constrained compute speeds and communication bandwidths other than energy consumption. There are quite a large amount of works focusing on reducing the communication cost to improve the efficiency of federated training. For example, LotteryFL [31] applies the lottery ticket hypothesis to learn a pruned lottery ticket network such that the communication cost between the server and clients is significantly reduced due to the compact size of model updates. Besides sparsifying/pruning the model updates, another approach to reducing communication costs is quantifying the model updates. As an example, AdaQuantFL [20] proposes an adaptive quantization strategy that changes the number of quantization levels during the course of federated training to reduce the communication cost while achieving a low error floor. Our work shares a similar objective on optimizing the system performance of FL; however, we tackle this problem from the perspective of client selection.

Client Selection in FL. Client selection has recently emerged as an essential problem in FL given its potential in addressing the scalability challenge in FL [49], its role as an alternative to asynchronous federated optimization to overcoming the adverse effects of stragglers [51], as well as its benefits on improving the performance and efficiency of federated training [8]. While simple client selection scheme such as random selection has been widely used, especially in the scenario where clients might only participate once in the entire training process, the majority of the newly proposed client selection methods focus on the scenario where most clients are stable and available to participate in most rounds of training and client-side state information is used as a feedback to guide the client selection [8]. For example, ClusterFL [41] exploits the intrinsic cluster structure among different clients based on their local data distributions. Based on this information, it drops straggling clients who converge slower than others or clients who are less related to other clients within each cluster. In doing so, there are fewer clients interacting with the server, so the overall communication time is reduced. The work closest to ours is Oort [27], which proposes to leverage both data and system heterogeneity and explores the sweet point between them with a unified client selection scheme. However, as explained and demonstrated in previous sections, Oort fails to fully exploit both the data and system efficiency due to its coarse-grained design. In contrast, PyramidFL effectively addresses its limitations and outperforms Oort across diverse FL tasks and datasets.

8 CONCLUSION

While today's FL efforts have been optimizing the statistical and system efficiency via specially designed client selection mechanisms, the state-of-the-arts fail to profile the utility function in a fine-grained manner, leading to sub-optimal time-to-accuracy performance. This paper presents PyramidFL to enable a fine-grained client selection for FL at scale. Specifically, the server defines a utility function in terms of the statistical efficiency (i.e., accuracy) and system efficiency (i.e., time) to rank all clients, then select top-K clients to participate in the next-round training. Moreover, the server calculates the participant-level importance ranking in terms of their statistical efficiency and how many training epoch a participant can use to see more data with noticing the per-round training time constraint, then share the information to make each participant can optimize their local training to provide the most competitive utility for the client selection in the future rounds. The utility optimization includes adaptive data epoch for enhancing its statistical efficiency and importance-based dropout for improving its system efficiency. Compared with the SOTA, PyramidFL consumes less wall clock time (speed up of $2.71\times-13.66\times$) while reaching a higher final accuracy/perplexity (optimize by $3.68\%-7.33\%$) for a variety of real-life FL applications with multiple ML models and optimizers.

9 ACKNOWLEDGEMENT

We sincerely thank the anonymous reviewers and our shepherd for their valuable feedback. This work was partially supported by NSF Award PFI:BIC-1632051 and CNS-1909177.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of ACM SIGSAC conference on computer and communications security.*

[2] Ahmed M Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A Fahmy. 2021. Resource-Efficient Federated Learning. *arXiv preprint arXiv:2111.01108* (2021).

[3] Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. 2016. Variance Reduction in SGD by Distributed Importance Sampling. *arXiv:1511.06481 [cs, stat]* (2016).

[4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *Proceedings of NeurIPS*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30.

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. In *Proceedings of MLSys.*

[6] Nader Bouacida, Jiahui Hou, Hui Zang, and Xin Liu. 2021. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. In *Proceedings of IEEE INFOCOM Workshops (INFOCOM WKSHPS).*

[7] Yae Jee Cho, Samarth Gupta, Gauri Joshi, and Osman Yağan. 2020. Bandit-based Communication-Efficient Client Selection Strategies for Federated Learning. In *Proceedings of IEEE Asilomar Conference on Signals, Systems, and Computers.*

[8] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. 2020. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv preprint arXiv:2010.01243* (2020).

[9] Stack Overflow Data. Retrieved by July 4th 2021. BigQuery public datasets. In *https://cloud.google.com/bigquery/public-data.*

[10] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* (2020).

[11] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[12] Jack Goetz, Kshitiz Malik, Duc Bui, Seungwhan Moon, Honglei Liu, and Anuj Kumar. 2019. Active federated learning. *arXiv preprint arXiv:1909.12641* (2019).

[13] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A Research Library and Benchmark for Federated Machine Learning. In *Conference on Neural Information Processing Systems (NeurIPS) Federated Learning Workshop.*

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of IEEE/CVF CVPR.*

[15] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos I Venieris, and Nicholas D Lane. 2021. FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout. *arXiv preprint arXiv:2102.13451* (2021).

[16] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia: Geo-distributed machine learning approaching {LAN} speeds. In *Proceedings of {USENIX} {NSDI}.*

[17] Junxian Huang, Cheng Chen, Yutong Pei, Zhaoguang Wang, Zhiyun Qian, Feng Qian, Birjodh Tiwana, Qiang Xu, Z Mao, Ming Zhang, et al. 2011. Mobiperf: Mobile network measurement system. *Technical Report. University of Michigan and Microsoft Research* (2011).

[18] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. 2019. Ai benchmark: All about deep learning on smartphones in 2019. In *Proceedings of IEEE/CVF International Conference on Computer Vision Workshop (ICCVW).*

[19] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. 2019. Communication-efficient distributed SGD with sketching. *arXiv preprint arXiv:1903.04488* (2019).

[20] Divyansh Jhunjhunwala, Advait Gadhikar, Gauri Joshi, and Yonina C Eldar. 2021. Adaptive quantization of model updates for communication-efficient federated learning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 3110–3114.

[21] Shaoxiong Ji, Wenqi Jiang, Anwar Walid, and Xue Li. 2020. Dynamic sampling and selective masking for communication-efficient federated learning. *arXiv preprint arXiv:2003.09603* (2020).

[22] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[23] Angelos Katharopoulos and François Fleuret. 2018. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of ICML.* PMLR.

[24] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[25] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. 2018. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982* (2018).

[26] Fan Lai, Yinwei Dai, Xiangfeng Zhu, and Mosharaf Chowdhury. 2021. FedScale: Benchmarking model and system performance of federated learning. *arXiv preprint arXiv:2105.11367* (2021).

[27] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient Federated Learning via Guided Participant Selection. In *Proceedings of USENIX OSDI.*

[28] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[29] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. 2019. Federated learning for keyword spotting. In *Proceedings of IEEE ICASSP.* 6341–6345.

[30] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. 2021. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *Proceedings of ACM MobiCom.*

[31] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2020. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371* (2020).

[32] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. Fed-Mask: Joint Computation and Communication-Efficient Personalized Federated Learning via Heterogeneous Masking. In *Proceedings of ACM SenSys.*

[33] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* (2020).

[34] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. In *Proceedings of MLSys.*

[35] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. 2019. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497* (2019).

[36] Bill Yuchen Lin, Chaoyang He, Zihang Zeng, Hulin Wang, Yufen Huang, Mahdi Soltanolkotabi, Xiang Ren, and Salman Avestimehr. 2021. FedNLP: Benchmarking Federated Learning Methods for Natural Language Processing Tasks. *arXiv preprint arXiv:2104.08815* (2021).

[37] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. 2020. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619* (2020).

[38] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics, PMLR.*

[39] Umberto Michieli and Mete Ozay. 2021. Are All Users Treated Fairly in Federated Learning Systems?. In *Proceedings of the IEEE/CVF CVPR.*

[40] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *Proceeding of IEEE International Conference on Communications (ICC).* IEEE, 1–7.

[41] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. ClusterFL: A Similarity-Aware Federated Learning System for Human Activity Recognition. In *Proceedings of ACM MobiSys.*

[42] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H Brendan McMahan. 2021. Adaptive federated optimization. In *Proceedings of ICLR.*

[43] Monica Ribero and Haris Vikalo. 2020. Communication-efficient federated learning via optimal client sampling. *arXiv preprint arXiv:2007.15197* (2020).

[44] Yichen Ruan, Xiaoxi Zhang, Shu-Che Liang, and Carlee Joe-Wong. 2021. Towards Flexible Device Participation in Federated Learning. In *Proceedings of International Conference on Artificial Intelligence and Statistics.* PMLR, 3403–3411.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF CVPR.*

[46] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. 2017. Federated multi-task learning. *arXiv preprint arXiv:1705.10467* (2017).

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* (2014).

[48] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. 2021. FedDL: Federated Learning via Dynamic Layer Sharing for Human Activity Recognition. In *Proceedings of ACM SenSys.*

[49] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Aguera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang

He, Samuel Horvath, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konecny, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtarik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. 2021. A Field Guide to Federated Optimization. arXiv:2107.06917 [cs.LG]

[50] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[51] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).

[52] Tianlong Yu, Tian Li, Yuqiong Sun, Susanta Nanda, Virginia Smith, Vyas Sekar, and Srinivasan Seshan. 2020. Learning context-aware policies from multiple smart homes via federated multi-task learning. In *Proceedings of IEEE/ACM International Conference on Internet-of-Things Design and Implementation (IoTDI)*.

[53] Xiao Zeng, Ming Yan, and Mi Zhang. 2021. Mercury: Efficient On-Device Distributed DNN Training via Stochastic Importance Sampling. In *Proceedings of ACM SenSys*.

[54] Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and Salman Avestimehr. 2021. Federated Learning for Internet of Things: Applications, Challenges, and Opportunities. arXiv:2111.07494 [cs.LG]

[55] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE/CVF CVPR*.

[56] Peilin Zhao and Tong Zhang. 2015. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of ICML*. PMLR.