Oort: Informed Participant Selection for Scalable Federated Learning

Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, Mosharaf Chowdhury
University of Michigan

Abstract

Federated Learning (FL) is an emerging direction in distributed machine learning (ML) that enables in-situ model training and testing on edge data. Despite having the same end goals as traditional ML, FL executions differ significantly in scale, spanning thousands to millions of participating devices. As a result, data characteristics and device capabilities vary widely across clients. Yet, existing efforts randomly select FL participants, which leads to poor model and system efficiency.

In this paper, we propose Oort to improve the performance of federated training and testing with guided participant selection. With an aim to improve time-to-accuracy performance in model training, Oort prioritizes the use of those clients who have both data that offers the greatest utility in improving model accuracy and the capability to run training quickly. To enable FL developers to interpret their results in model testing, Oort enforces their requirements on the distribution of participant data while improving the duration of federated testing by cherry-picking clients. Our evaluation shows that, compared to existing participant selection mechanisms, Oort improves time-to-accuracy performance by 1.2×-14.1× and final model accuracy by 1.3%-9.8%, while efficiently enforcing developer requirements on data distributions at the scale of millions of clients.

1 Introduction

Machine learning (ML) today is experiencing a paradigm shift from cloud datacenters toward the edge [20, 44]. Edge devices, ranging from smartphones and laptops to enterprise surveillance cameras and edge clusters, routinely store application data and provide the foundation for machine learning beyond datacenters. With the goal of not exposing raw data, large companies such as Google and Apple deploy *federated learning (FL)* for computer vision (CV) and natural language processing (NLP) tasks across user devices [2, 26, 35, 76]; NVIDIA applies FL to create medical imaging AI [51]; smart cities perform in-situ image training and testing on AI cameras to avoid expensive data migration [37, 42]; and video streaming and networking communities use FL to interpret and react to network conditions [10, 74].

Although the life cycle of an FL model is similar to that in traditional ML at a high level (§2.1), the underlying execution in FL is spread across thousands to millions of devices in the wild. Similar to traditional ML, the wall clock time for training a model to reach an accuracy target (i.e., time-to-accuracy) is still a key performance objective even though it may take significantly longer [44]. In addition, to inspect these models being trained, to perform hyperparameter tuning of models prototyped with proxy data [58], or to validate deployed models [73, 74], developers may want to perform federated model testing on edge devices, wherein a representative categorical distribution[1] of the testing set is often required [22, 66] (§2.1).

Unfortunately, clients may not all be simultaneously available for FL training or testing [20]; they may have heterogeneous data distributions and system capabilities [28]; and including too many may lead to wasted work and suboptimal performance [20] (§2). Consequently, a fundamental problem in practical FL is the *selection of a "good" subset of clients as participants*, where each participant locally processes its own data, and only their results are collected and aggregated at a (logically) centralized coordinator.

Existing works optimize for *statistical model efficiency* (i.e., better training accuracy with fewer training rounds) [23, 49, 60, 70] or *system efficiency* (i.e., shorter rounds) [55, 69], while randomly selecting participants. Unfortunately, random participant selection results in poor performance of federated training because of large heterogeneity in device speed and/or data characteristics (§2.3). More subtly, because they randomly select participants, existing FL efforts can lead to biased testing accuracy and loss of confidence in results (§2.3). As a result, developers often resort to more participants than perhaps needed [58, 71].

We present Oort to enable FL developers to guide participant selection for their models (§3). It provides better time-to-accuracy performance for federated model training by prioritizing certain participants and enforces developer-specified data distribution requirements during model testing. Oort makes informed participant selection by relying on the information available in today's FL [44] and offers its benefits with little modification to existing efforts.

Selecting participants for federated model training is non-trivial because of the wide variance in system and statistical model utility both across clients and of any specific client over time (as the trained model changes). For a large population, capturing the latest utility of all clients is impractical. To this end, we identify clients with high utility, which is measured in terms of their most recent aggregated training loss,

[1] A categorical distribution is a discrete probability distribution that describes the possible results of a random variable that can take on one of K possible categories (e.g., unique labels).

adjusted for spatiotemporal variations. We then employ an online exploration-exploitation strategy to pick out a high-utility client pool and probabilistically select participants among it, while ensuring robustness to outliers and respecting privacy (§4.3). Moreover, we penalize those clients whose speed is likely to elongate the duration necessary to complete global aggregation, and adaptively allow for longer training rounds in order to accommodate clients with higher data utility (§4).

Although FL developers often have well-defined requirements on their testing data (e.g., a number of samples for each category or following the representative categorical distribution), satisfying these requirements is not straightforward even with complete information of clients' data characteristics [74]. This appears to be a classical bin covering problem, where a subset of data bins (i.e., participants) are selected to cover the requested quantity of data. At scale, solving this problem is infeasible due to high computational complexity. We propose a greedy heuristic that can efficiently serve the developer's data requirements even at the scale of millions of clients, while optimizing the duration of model testing. The challenge is exacerbated for private FL scenarios where clients' data characteristics are unknown [31]. We cap the deviation of participant data from the global distribution using a provable estimation mechanism [17], and perform selection by bounding the number of participants needed (§5).

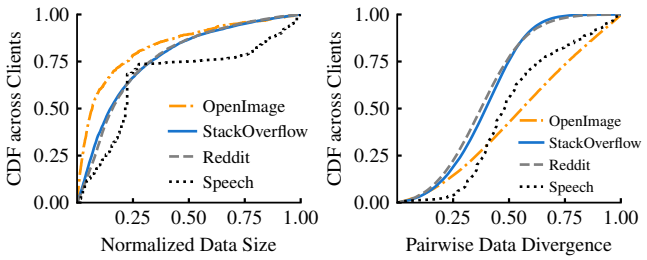
Our evaluation of Oort, across different scales of real-world workloads for CV and NLP tasks, shows that Oort can improve the performance of FL training and testing, while efficiently satisfying developer requirements on the data distribution (§7). Compared to the state-of-the-art selection techniques used in solutions like YoGi [63] and Prox [49], Oort improves the time-to-accuracy performance of federated training by 1.2×-14.1×, while achieving 1.3%-9.8% better final model accuracy. For federated testing, Oort can efficiently respond to developer-specified data distribution on thousands of categories across millions of clients, and it improves testing duration by 4.7× on average over state-of-the-art solutions.

2 Background and Motivation

We start with a quick primer on federated learning (§2.1), followed by the challenges it faces based on our analysis of real-world datasets (§2.2). Next, we highlight the key shortcomings of the state-of-the-art that motivate our work (§2.3).

2.1 Federated Learning

Federated learning is an emerging ML paradigm, where many distributed participants collectively process the raw data on their devices [11]. A (logically) centralized coordinator manages the aggregation of results from participants [13, 20]. Similar to traditional ML, an FL developer first prototypes model architectures and hyperparameters with a proxy dataset. After selecting a suitable configuration, she can use federated training to improve model performance by training across a crowd of participants [20]. Later, she can employ federated



(a) Unbalanced data size. (b) Heterogeneous data distribution.

Figure 1: ***Client data differs in size and distribution greatly.***

testing to evaluate the ground-truth of model performance on end-user data. While training and testing can take place in the same FL application, they pose different challenges.

Federated model training. Federated model training aims to learn an accurate model across thousands to potentially millions of clients. Because of the large population size and diversity of user data and their devices in FL, training runs on a subset of clients (hundreds of participants) in each round, and often takes hundreds of rounds – each round lasts a few minutes – and several days to complete. For example, in its Gboard keyboard, Google runs federated training of NLP models over weeks across 1.5 million end devices [4, 76]. Achieving a target model accuracy with less wall clock time (i.e., time-to-accuracy) is often the primary target of training.

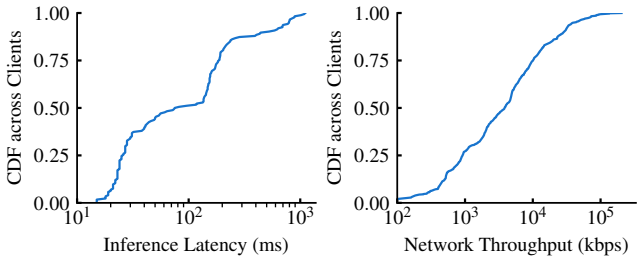
Federated model testing. FL developers sometimes test their model's performance to fine-tune model configurations interactively with real-life datasets [48], to inspect the model accuracy along training to detect cut-off accuracy, or to validate the pre-trained model before deployment [71,74]. Similar to traditional model testing, developers often request the representativeness of the testing set with requirements like *"50k representative samples"* [25], or the exact requirement on each category like *"x samples of class y"* to investigate model performance on specific categories [62]. When the data characteristics of participants are not available, coarse-grained yet non-trivial requests, such as *"a subset with less than X% data deviation from the global"* are still informative [27,54,58].

2.2 Challenges in Federated Learning

Apart from the challenges faced in traditional ML, FL introduces new challenges in terms of data, systems, and privacy.

Heterogeneous statistical data. Data in each FL participant is typically generated in a distributed manner under different contexts and stored independently. For example, images collected by cameras will reflect the demographics of each camera's location. This breaks down the widely-accepted assumption in traditional ML that samples are independent and identically distributed (i.i.d.) from a data distribution.

We analyze four real-world datasets for CV (OpenImage [3]) and NLP (StackOverflow [9], Reddit [8] and Google Speech [72]) tasks. Each consists of thousands or up to millions of clients and millions of data points (details in Ap-



(a) Heterogeneous compute capacity. (b) Heterogeneous network capacity.

Figure 2: ***Client system performance differs significantly.***

pendix A). In each individual dataset, we see a high statistical deviation across clients not only in the quantity of samples (Figure 1(a)) but also in the data distribution (Figure 1(b)).[2]

Heterogeneous system performance. As individual data samples are tightly coupled with the participant device, in-situ computation on this data experiences significant heterogeneity in system performance. We analyze the inference latency of MobileNet [65] across hundreds of mobile phones used in a real-world FL deployment [76], and their available bandwidth (details in Appendix A). Unlike the homogeneous setting in datacenter ML, system performance across clients exhibits an order-of-magnitude difference in both computational capabilities (Figure 2(a)) and network bandwidth (Figure 2(b)).

Enormous population and pervasive uncertainty. While traditional ML runs in a well-managed cluster with a number of machines, federated learning often involves up to millions of clients, which makes it challenging for the coordinator to efficiently identify and manage valuable participants. Moreover, during execution, participants often vary in system performance [20, 44] – they may slow down, drop out, or rejoin – leading to uncertainties in the availability of data too.

Privacy concerns. Inquiring the privacy-sensitive information of clients (e.g., raw data or even data distribution) can alienate participants in contributing to FL [26,67,68]. Without collecting the raw data, how to select participants whose data is of high utility in federated training? When the data distribution is unavailable, how to select a representative participant dataset for federated testing?

2.3 Limitations of Existing FL Solutions

Recent efforts have made considerable progress in tackling some of the above challenges (§8), but they leave large room for improvements.

Suboptimality in maximizing efficiency. Existing FL solutions mostly rely on hindsight – given a pool of participants, they optimize model performance [50, 60] or system efficiency [55] to tackle data and system heterogeneity. However, the potential for curbing these disadvantages by cherry-picking participants before execution has largely been over-

[2]We measure the pairwise deviation of categorical distributions between two clients, using the popular L1-divergence metric [56].

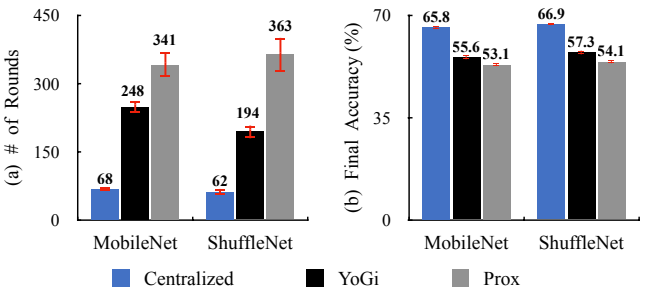
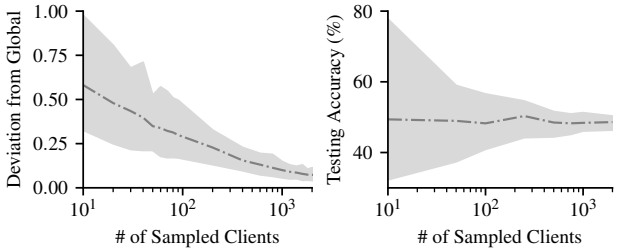


Figure 3: ***Existing works are suboptimal in: (a) round-to-accuracy performance and (b) final model accuracy. (a) reports number of rounds required to reach the highest accuracy of Prox on MobileNet (i.e., 53.1%). Error bars show standard deviation.***



(a) Data deviation vs. participant size. (b) Accuracy vs. participant size.

Figure 4: ***Participant selection today leads to (a) deviations from developer requirements, and thus (b) affects testing result, unless a large number of participants are selected. Shadow in each figure indicates the [min, max] range of the y-axis values over 1000 runs given the same x-axis input, and each line reports the median.***

looked. For example, existing federated training still relies on randomly picking participants in each training round [20], which is suboptimal in system and statistical model efficiency.

To quantify the performance loss, we train two popular image classification models tailored for mobile devices (i.e., MobileNet [65] and ShuffleNet [77]) with 1.4 million images of the OpenImage dataset, and randomly pick 100 participants out of more than 14k clients in each training round. We consider a performance *upper bound* by creating a hypothetical centralized case where images are evenly distributed across only 100 clients, and train on all 100 clients in each round. As shown in Figure 3, even with state-of-the-art optimizations, such as YoGi [63] and Prox [49],[3] the round-to-accuracy and final model accuracy are both far from the upper-bound. Note that system heterogeneity will further exacerbate the suboptimality of time-to-accuracy performance.

Inability to enforce developer requirements. While a FL developer often fine-tunes her model by understanding the input dataset, existing solutions do not provide any mechanism for the developer to reason about what data her FL model was trained or tested on. Even worse, existing participant selection can lead to bias and loss of confidence in FL results [22, 39].

[3]These two adapt traditional stochastic gradient descent algorithms to tackle the heterogeneity of the client datasets.

To better understand how existing works fall short, we take the global categorical distribution as an example requirement, and experiment with the above pre-trained ShuffleNet model. Figure 4(a) shows that: (i) even for the same number of participants, random selection can result in noticeable data deviations from the target distribution; (ii) while this deviation decreases as more participants are involved, it is non-trivial to quantify how it varies with different number of participants, even if we ignore the cost of enlarging the participant set. One natural effect of violating developer specification is bias in results (Figure 4(b)), where we test the accuracy of the same model on these participants. We observe that a biased testing set results in high uncertainties in testing accuracy.

3 Oort Overview

Existing participant selection mechanisms [49, 63] in FL underperform for federated model training and testing because of client heterogeneity. Oort improves performance by judiciously selecting participants while enforcing FL developers' requirements on data distribution. In this section, we provide an overview of how Oort fits in the bigger picture to help the reader follow its primary contributions.

3.1 Architecture

Oort is a participant selection framework that identifies and cherry-picks valuable participants for FL training and testing. It is located inside the coordinator of an FL framework and interacts with the driver of an FL execution (e.g., PySyft [7]). Given the developer-specific requirements and collected feedbacks from the driver, it responds with a list of participants, whereas the driver is in charge of initiating and managing execution on the Oort-selected remote participants.

Figure 5 shows how Oort interacts with the developer and FL execution frameworks. ① *Job submission*: the developer submits and specifies the participant selection criteria to the FL coordinator in the cloud. ② *Participant selection*: the coordinator enquires the clients meeting eligibility properties (e.g., battery level), and forwards their characteristics (e.g., liveness) to Oort. Given the developer requirements (and execution feedbacks in case of training ②a), Oort selects participants based on the given criteria and notifies the coordinator of this participant selection (②b). ③ *Execution*: the coordinator distributes relevant profiles (e.g., model) to these participants, and then each participant independently computes results (e.g., model weights in training) on her data; ④ *Aggregation*: when participants complete the computation, the coordinator aggregates updates from participants.

During federated training, where the coordinator initiates the next training round after aggregating updates from enough number of participants [20], it iterates over ②-④ in each round until a certain accuracy target has been reached. In contrast, federated model testing is a single-round execution, which can be used in conjunction with training to determine the currently attained accuracy or independently to test the deployed model.

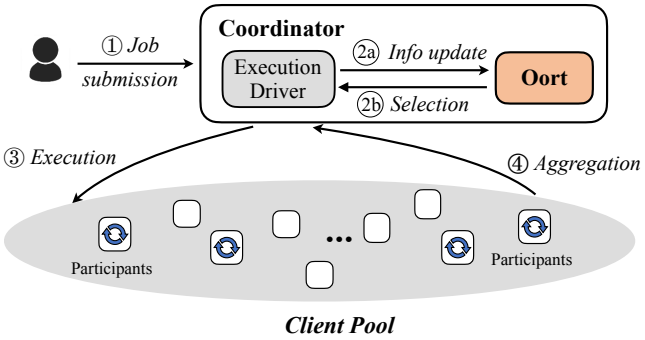


Figure 5: ***Oort architecture. The driver of the FL framework interacts with Oort using a client library.***

3.2 Oort Interface

Oort employs two distinct selectors that developers can access via a client library during FL training and testing.

Training selector. This selector aims to improve the time-to-accuracy performance of federated training. To this end, it captures the utility of clients in training, and efficiently explores and selects high-utility clients at runtime.

```
1  import Oort
2
3  def federated_model_training():
4    selector = Oort.create_training_selector(config)
5
6    # Train to target testing accuracy
7    while (federated_model_testing() < target):
8      # Collect feedbacks of last round
9      feedbacks = engine.get_participant_feedback()
10
11     # Update the utility of clients
12     for clientId in feedbacks:
13       selector.update_client_util(
14                   clientId, feedbacks[clientId])
15
16     # Pick 100 high-utility participants
17     participants = selector.select_participant(100)
18
19     ... # Activate training on remote clients
```

Figure 6: ***Code snippet of Oort interaction during FL training.***

Figure 6 presents an example of how FL developers and frameworks interact with Oort during training. In each training round, Oort collects feedbacks from the engine driver, and updates the utility of individual clients (Line 12-14). Thereafter, it cherry-picks high-utility clients to feed the underlying execution (Line 17). We elaborate more on client utility and the selection mechanism in Section 4.

Testing selector. When client data characteristics (e.g., categorical distribution) are available, the testing selector cherry-picks participants to serve the exact developer-specified requirements on data. Otherwise, it caps the data deviation of participants by determining the number of participants needed. Oort currently supports two types of selection criteria. We explain both in Section 5 along with corresponding examples.

4 Federated Model Training

Next, we describe how Oort quantifies the client utility in federated training (§4.1), how it selects high-utility clients at

scale despite being unable to maintain the latest utility value for every client as the model evolves over time (§4.2), and how to enable selection while respecting privacy (§4.3).

4.1 Client Utility

Time-to-accuracy performance of FL training relies on two aspects: (i) the number of rounds taken to reach target accuracy, which we refer to as *statistical efficiency*; and (ii) the duration of each training round, which we refer to as *system efficiency*. Oort associates with every client its *utility* in optimizing either form of efficiency. The data stored on the client and the speed with which it can process data determine its utility with respect to statistical and system efficiency, which we respectively refer to as statistical and system utility.

Capture statistical utility with negligible overhead. We first model the statistical utility of clients. Say client i has a bin B_i of training samples locally stored. Then, to improve the round-to-accuracy performance via importance sampling for ML [46,78], the oracle is to pick bin B_i with a probability proportional to its importance $|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} \| \nabla f(k) \|^2}$, where $\| \nabla f(k) \|$ is the L2-norm of the unique sample k's gradient $\nabla f(k)$ in bin B_i. Intuitively, this means selecting the bin with larger accumulated gradient norm across all of its samples.

While importance sampling has been theoretically proven to outperform random sampling [32], simply taking this importance as the statistical utility is impractical as it requires an extra time-consuming pass over the client data to generate the gradient norm of each sample.[4] Even worse, this gradient norm varies as the model updates over time.

To avoid extra cost, we introduce a pragmatic approximation of statistical utility instead. At the core of model training, training loss measures the estimation error between model predictions and the ground truth, wherein the model minimizes this loss by iteratively taking the derivative (i.e., gradient) of training loss with respect to current weights. Our insight is that a larger gradient norm often attributes to a bigger loss given the same model weights [43]. Therefore, we define the statistical utility $U(i)$ of client i as $U(i) = |B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}$, where the training loss $Loss(k)$ of sample k is automatically generated during training with negligible collection overhead. As such, we consider clients that currently accumulate a bigger loss to be more important for future rounds.

Our statistical utility can capture the heterogeneous data utility across and within categories and samples, as training loss can measure the prediction confidence of individual samples even when the prediction is correct [27]. We present the theoretical proof for its effectiveness in Appendix B, and empirically show its close-to-optimal performance (§7.2.2).

Strike a trade-off between statistical and system efficiency in aggregations. Updating the global model requires the aggregation of participants. Therefore, using a client with high statistical utility may hamper the speed of a training round if that client happens to be the bottleneck in aggregation. This motivates us to jointly consider statistical and system utility while choosing participants.

To strike a good trade-off, we aim at maximizing the statistical utility of participants without greatly sacrificing the system efficiency of each training round. To this end, we extend the utility of client i by incorporating a global system utility in terms of the duration of each training round:

$$Util(i) = \underbrace{|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} Loss(k)^2}}_{Statistical\ utility\ U(i)} \times \underbrace{(\frac{T}{t_i})^{\mathbb{1}(T<t_i)\times\alpha}}_{Global\ sys\ utility} \quad (1)$$

where T is the expected duration of each round, t_i is the amount of time that client i takes to process its samples ($=\frac{|B_i|}{c_i}$, where c_i is the system speed that can be measured from previous rounds by the coordinator), $\mathbb{1}(x)$ is an indicator function that takes value 1 if x is true and 0 otherwise, and α is the developer-specified penalty for stragglers, which enables us to marginalize participants that bottleneck the desired speed. We do not reward the non-straggler clients because their completions do not impact the overall duration of a training round.

This formulation assumes that all samples at a client are processed in that training round. Even if the estimated t_i for a client is greater than the desired round duration T, Oort might pick that client if the statistical utility outweighs its slow speed. Alternatively, if the developer wishes to cap every round at a certain duration or after a certain number of steps [55], then either only clients with $t_i < T$ can be considered or a subset of a participant's samples can be processed [49,63].

4.2 Adaptive Participant Selection

Given the above definition of client utility, we need to address the following challenges in order to select participants with the highest utility in each training round.

- First, a client's utility can only be determined after it has participated in at least one training round; how to choose from clients at scale without having to try all clients once?
- Second, since not every client participates in every round, how to account for the change in a client's utility since when it was last a participant?
- Third, how to be robust to outliers in the presence of corrupted clients (e.g., with noisy data)?
- Lastly, how to pick the duration of every round?

To tackle these challenges, we introduce an exploration-exploitation strategy for participant selection (Algorithm 1).

Online exploration-exploitation of high-utility clients. Selecting participants out of numerous clients can be modeled as an instance of the multi-armed bandit problem, where each client is an "arm" of the bandit, and the utility obtained is the "reward." The goal is to maximize the long-term reward by

[4] The model generates the training loss of each sample during training, but calculates the gradient of the mini-batch instead of individual samples.

Input: Client set $\mathbb{C}$, sample size K, exploitation factor ε, pacer step Δ, step window W, penalty α
Output: Participant set $\mathbb{P}$

/* Initialize global variables. */
1 $\mathbb{E} \leftarrow \emptyset$; $\mathbb{U} \leftarrow \emptyset$ ▷ Explored clients and statistical utility.
2 $\mathbb{L} \leftarrow \emptyset$; $\mathbb{D} \leftarrow \emptyset$ ▷ Last involved round and duration.
3 $R \leftarrow 0$; $T \leftarrow \Delta$ ▷ Round counter and preferred round duration.

/* Participant selection for each round. */
4 **Function** SelectParticipant ($\mathbb{C}$, K, ε, T, α)
5 $Util \leftarrow \emptyset$; $R \leftarrow R+1$

/* Update and clip the feedback; blacklist outliers. */
6 UpdateWithFeedback($\mathbb{E}$, $\mathbb{U}$, $\mathbb{L}$, $\mathbb{D}$)

/* Pacer: Relaxes global system preference T if the statistical utility achieved decreases in last W rounds. */
7 **if** $\sum \mathbb{U}(R-2W:R-W) > \sum \mathbb{U}(R-W:R)$ **then**
8 $T \leftarrow T+\Delta$

/* Exploitation #1: Calculate client utility. */
9 **for** *client* $i \in \mathbb{E}$ **do**
10 $Util(i) \leftarrow \mathbb{U}(i) + \sqrt{\frac{0.1 \log R}{\mathbb{L}(i)}}$ ▷ Temporal uncertainty.
11 **if** $T < \mathbb{D}(i)$ **then** ▷ Global system utility.
12 $Util(i) \leftarrow Util(i) \times (\frac{T}{\mathbb{D}(i)})^{\alpha}$

/* Exploitation #2: admit clients with greater than c% of cut-off utility, and then sample $\varepsilon \times K$ clients by utility. */
13 $Util \leftarrow$ SortAsc(Util)
14 $\mathbb{W} \leftarrow$ CutOffUtil($\mathbb{E}$, $c \times Util(\varepsilon \times K)$)
15 $\mathbb{P} \leftarrow$ SampleByUtil($\mathbb{W}$, $Util$, $\varepsilon \times K$)

/* Exploration: sample unexplored clients by speed. */
16 $\mathbb{P} \leftarrow \mathbb{P} \cup$ SampleBySpeed($\mathbb{C}-\mathbb{E}$, $(1-\varepsilon) \times K$)
17 **return** $\mathbb{P}$

Alg. 1: Participant selection w/ exploration-exploitation.

balancing the exploration and exploitation of different arms based on the previous observations on arms and rewards [15].

The Upper Confidence Bounds (UCB) algorithm is widely used to solve the scalable multi-armed bandit problem [15,41]. It measures the potential of selecting the arm by an upper confidence bound of the underlying reward. This upper bound is formulated as the sum of the observed reward and its uncertainty, wherein a larger number of trials of the arm implies a smaller uncertainty. At runtime, the UCB algorithm selects the arm with the maximum upper confidence bound.

Similar to the UCB algorithm, Oort efficiently explores potential participants under spatial variation, while intelligently exploiting observed high-utility participants under temporal variation.[5] At the beginning of each selection round, Oort receives the feedback of the last training round, and updates the statistical utility and system performance of clients (Line 6). For the explored clients, Oort calculates their client utility and narrows down the selection by exploiting the high-utility participants (Line 9-15). Meanwhile, Oort samples $\varepsilon(\in [0, 1])$ fraction of participants to explore potential participants that had not been selected before (Line 16), which turns to full exploration without replacement as $\varepsilon \rightarrow 0$. Although we cannot learn the statistical utility of not-yet-tried clients, one can decide to prioritize the unexplored clients with faster system speed when possible (e.g., by inferring from device models), instead of performing random exploration (Line 16).

Exploitation under uncertainties in client utility. Oort exploits high-utility clients adaptively while being staleness-aware as follows. First, motivated by the confidence interval used to measure the uncertainty in reward, we introduce an incentive term, which shares the same shape of the confidence interval in UCB [41], to account for the staleness (Line 10), whereby we gradually increase the utility of a client if it has been overlooked for a long time. Moreover, instead of simply picking clients with top-k utility, we allow a confidence interval c on the cut-off utility (95% by default in Line 13-14). Specifically, we admit clients whose utility is greater than the c% of the top $(\varepsilon \times K)$-th participant. Among this high-utility client pool, Oort samples participants with probability proportional to their utility (Line 15). This adaptive exploitation mitigates the approximation bias by prioritizing participants opportunistically while preserving a high quality as a whole.

Robust exploitation under outliers. Simply prioritizing high utility clients can be vulnerable to outliers in unfavorable settings. For example, corrupted clients may have noisy data, leading to high training loss, or even report arbitrarily high training loss intentionally. For robustness, we (i) remove the client in selection after she has been picked over a given number of rounds. This removes the perceived outliers in terms of participation (Line 6); (ii) clip the statistical utility of a client by capping her utility to no more than an upper bound (e.g., 95% value in utility distributions), and sample clients probabilistically among the high-utility client pool (Line 15). As such, the chance of selecting outliers is significantly decreased under the scale of clients in FL. We show that Oort outperforms existing mechanisms while being robust (§7.2.3).

Online selection with a tradeoff-aware pacer. Determining the preferred round duration T in Equation (1), which strikes the trade-off between the statistical and system efficiency in aggregations, can be challenging. Indeed, the total statistical utility achieved (i.e., $\sum U(i)$) can decrease round by round, because the training loss decreases as the model improves over rounds. If we persist in suppressing clients

[5] We borrow from the UCB algorithm because, in contrast to sophisticated models (e.g., reinforcement learning), it is scalable and flexible even when the solution space (e.g., number of clients) varies dramatically over time.

with high statistical utility but low system speed (Line 12), the model may converge to suboptimal accuracy (§7.2.2).

While the developer can specify her preferred T, Oort employs a pacer to determine this value at runtime (Line 7). The intuition is that, when the accumulated statistical utility in the past W rounds decreases, the pacer relaxes this system constraint T to bargain with the statistical efficiency again.

4.3 Privacy Concern in Feedback Collection

A client's utility is a combination of its system speed and aggregated training loss. Today's FL coordinator already collects the former from past rounds.[6] Training loss measures the prediction uncertainties of a model without revealing the raw data and is often collected in real FL deployments too [35, 76].

We respect privacy further in three ways. First, Oort only relies on every client's *aggregated* training loss, which is computed locally by the client across *all* of its samples without revealing the loss distribution of individual samples. Second, when even the aggregated loss raises a privacy concern, Oort's probabilistic sampling of high utility clients makes it amenable for clients to add noise to the loss values they upload, like in existing differentially private FL [31]. Third, our design of Oort can flexibly accommodate other definitions of utility (e.g., using gradient norm of batches). We provide detailed analyses of each strategy in Appendix D to show that Oort can respect privacy while improving performance, both theoretically and empirically.

5 Federated Model Testing

In contrast to model training, enforcing stringent developer-defined requirements on data distribution is a first-order goal in model testing, which otherwise can lead to biased testing accuracy (§2.3). However, selecting the "right" set of participants is challenging due to the large client population size. Even worse, in many privacy-constrained scenarios, learning the individual data characteristics (e.g., categorical distribution) is prohibitively expensive or even prohibited [29, 64].

In this section, we describe how Oort navigates the tussle between data characteristics and system performance by deliberatively selecting participants. We first consider the case when individual data characteristics are known (§5.1). Next we show that, even without individual information, Oort can still perform judicious selection (§5.2).

5.1 Clairvoyant Participant Selection

When the individual data characteristics are provided, Oort can enforce the exact preference on specific categorical distribution formed by all participants, and improve the duration of model testing jobs by cherry-picking participants.

Problem formulation and design challenges. We start by casting this participant selection into a multi-dimensional bin covering problem. For each category $i(\in I)$ of interest, the developer has preference p_i (i.e., preference constraint), and an upper limit B (referred to as budget) on how many participants she can have [16].[7] Each participant $n(\in N)$ can contribute n_i samples out of its capacity $c_n{}^i$ on category i (i.e., capacity constraint). Given the compute speed s_n of participant n, the available bandwidth b_n and the size of data transfers d_n, we aim to minimize the duration of model testing:

$$\min \left\{ \max_{n\in N} \left(\frac{\sum_{i\in I} n_i}{s_n} + \frac{d_n}{b_n} \right) \right\} \quad \triangleright \text{ Minimize duration}$$

$$\text{s.t.} \quad \forall i \in I, \sum_{n\in N} n_i = p_i \quad \triangleright \text{ Preference Constraint}$$

$$\forall i \in I, \forall n \in N, n_i \le c_n^i \quad \triangleright \text{ Capacity Constraint}$$

$$\forall i \in I, \sum_{n\in N} \mathbb{1}(n_i > 0) \le B \quad \triangleright \text{ Budget Constraint}$$

The max-min formulation stems from the fact that model testing completes after aggregating results from the last participant. While this mixed-integer linear programming (MILP) model provides high-quality solutions, it has prohibitively high computational complexity for a large number of clients.

Scalable participant selection. We present a greedy heuristic with better scalability properties. Our key insight is to scale down the search space of MILP by tackling the preference constraint and job duration in isolation: we can first group a subset of feasible clients to satisfy the first-order preference constraint, and then optimize job duration with our MILP among this subset of clients.

To this end, we first iteratively add to our subset the client which has the most number of samples across all not-yet-satisfied categories, and deduct the preference constraint on each category by the corresponding capacity of this client. We stop this greedy grouping until the preference is met, or request a new budget if we exceed the budget B. Then, we apply a simplified MILP, where we remove the budget constraint and reduce the search space of clients from N to no more than B, to optimize the job duration.

Note that, to tolerate runtime stragglers or failures, the developer can admit more samples and/or clients with the quota of oversampling she wants in using our selector.

5.2 Non-Clairvoyant Participant Selection

Without knowing the individual data characteristics, the developer struggles to determine the number of participants she needs to be confident in the testing set's representativeness. As a result, she is inclined to be conservative and selects many participants, because selecting too few can lead to a biased testing set [58] (§2.3). However, admitting too many participants may exceed the budget and/or take too long because of the system heterogeneity. To enable developer-guided participant selection, Oort captures how the representativeness of testing set varies with different numbers of participants

[6] We only care whether a client can complete by the expected duration T. So, a client can even mask its precise speed by deferring its report.

[7] $B \to \inf$ represents a hypothetical case where FL can run on all clients.

and outputs the number of participants needed to guarantee a developer-provided deviation target.

Representativeness of data. We consider the deviation of the selected data group from the global dataset in the form of L1-distance, since many other distance metrics can be derived from L1-distance, such as L2 and chi-square distance [57].

For category X, its L1-distance ($|\bar{X} - E[\bar{X}]|$) captures how the average number of samples of selected participants (i.e., empirical value $\bar{X}$) deviates from that of all clients (i.e., expectation $E[\bar{X}]$). Note that the number of samples that one client holds is independent across different clients.[8] As such, for each client n, its number of samples X_n can be taken as a random instance of variable X. Given the tolerance ε on data deviation and confidence interval δ, our goal is to estimate the number of participants needed such that the deviation from the representative is bounded (i.e., $Pr[|\bar{X} - E[\bar{X}]| < \varepsilon] > \delta$).

With this model, we sample a certain number of participants and expect the categorical distribution they form is representative enough. This translates to a traditional problem of sampling stochastic variables, where a number of instances (i.e., participants) of X are sampled without replacement. As such, we apply existing solutions (e.g., Hoeffding-Serfling Bound [17]) to capture how their empirical value (i.e., data distribution formed by participants) deviates from their expectation (i.e., the global dataset) as the number of instances varies. Detailed results and proof are available in Appendix C.

Estimating the number of participants to cap deviation. Even when the individual data characteristics are not available, the developer can specify their tolerance on the deviation of the categorical distribution and preferred confidence interval (95% by default [57]), whereby Oort outputs the number of participants needed to preserve this preference. To use our model, the developer needs to input the range (i.e., maximum - minimum) of the number of samples that a client can hold. Learning this global information securely is well-established [24, 64], and the developer can assume a plausible limit (e.g., according to the capacity of device models) too.

Our model does not require any collection of the distribution of global or participant data, while guaranteeing the deviation target by bounding the number of participants needed. As a straw-man participant selection mechanism, the developer can randomly distribute her model to this Oort-determined number of participants, and adaptively select more participants to mitigate failures or stragglers. After collecting results from this number of participants, the developer can confirm the representativeness of computed data.

5.3 Usage Examples

Figure 7 shows example code snippets for both scenarios.

With the individual data characteristics, the developer can use the clairvoyant selector by dumping the client information

[8] The number of samples that one client holds will not be affected by the selection of any other clients at that time.

```python
import Oort

def federated_model_testing():
  selector = Oort.create_testing_selector()

  # [Clairvoyant] Update client data and sys info.
  for clientId in available_clients:
    selector.update_client_info(
                clientId, client_info[clientId])

  ''' [Clairvoyant Query] Give me 10k representative
      samples given data transfer size and budget'''
  participants = selector.select_representative(
                  10000, data_transfer_size=size,
                  budget=budget)

  ''' [Clairvoyant Query] Give me 5k samples of
     category A, 5k samples of category B'''
  participants = selector.select_by_category(
                  target_cate={'A': 5000, 'B': 5000},
                  data_transfer_size=size,
                  budget=budget)

  ''' [Non-Clairvoyant Query] Give me a subset w/
      less than 10 deviation from the global'''
  participants = selector.select_by_deviation(
                  dev_tolerance=10, max_min_range=500)

  ... # Activate execution and return accuracy
```

Figure 7: ***Code snippets of FL testing support in Oort.***

to Oort. Note that we use stratified sampling [56] to determine the number of samples in each category for queries like "*5k representative samples*". For the non-clairvoyant scenario, Oort receives the developer-specified deviation tolerance and the range of number of samples that a client can hold.

6 Implementation

We have implemented Oort as a Python library with around 2k lines of code. Oort provides simple APIs to abstract away the problem of participant selection, and developers can import Oort in their application codebase and interact with FL engines (e.g., PySyft [7] or TensorFlow Federated [11]).

Oort operates on and updates the client metadata (e.g., data distribution or system performance) fed by the FL developer and execution feedbacks at runtime. The metadata of each client in Oort is an object with small memory footprint. Oort caches these objects in memory during executions and periodically backs up them to persistent storage. We employ Gurobi solver [5] to solve the MILP in clairvoyant testing.

7 Evaluation

We evaluate Oort's effectiveness for four different ML models on four CV and NLP datasets.[9] We organize our evaluation by the FL activities with the following key results.

FL training results summary:

- Oort outperforms existing random participant selection by 1.2×-14.1× in time-to-accuracy performance, while achieving 1.3%-9.8% better final model accuracy (§7.2.1).
- Oort achieves close-to-optimal model efficiency by adaptively striking the trade-off between statistical and system

[9] We will make Oort implementation and the workloads open-source.

Task	Dataset	Accuracy Target	Model	Speedup for Prox [49]			Speedup for YoGi [63]		
				Stats.	Sys.	Overall	Stats.	Sys.	Overall
Image Classification	OpenImage [3]	53.1%	MobileNet [65]	4.2×	3.1×	13.0×	2.3×	1.5×	3.3×
			ShuffleNet [77]	4.8×	2.9×	14.1×	1.8×	3.2×	5.8×
Language Modeling	Reddit [8]	39 perplexity	Albert [47]	1.3×	6.4×	8.4×	1.5×	4.9×	7.3×
	StackOverflow [9]	39 perplexity	Albert [47]	2.1×	4.3×	9.1×	1.8×	4.4×	7.8×
Speech Recognition	Google Speech [72]	62.2%	ResNet-34 [36]	1.1×	1.1×	1.2×	1.2×	1.1×	1.3×

Table 1: ***Summary of improvements on time to accuracy. We tease apart the overall improvement with statistical and system ones, and take the highest accuracy that Prox can achieve as the target, which is moderate due to the high task complexity and lightweight models.***

efficiency with different components (§7.2.2).

- Oort outperforms its counterpart over a wide range of parameters and different scales of experiments, while being robust to outliers (§7.2.3).

FL testing results summary:

- For clairvoyant testing, Oort improves the testing duration by 4.7× w.r.t. Mixed Integer Linear Programming (MILP) solver on average, and is able to efficiently serve developer preferences across millions of clients (§7.3.1).
- For non-clairvoyant testing, Oort can preserve developer requirements on data deviation while reducing costs by bounding the number of participants needed (§7.3.2).

7.1 Methodology

Experimental setup. Oort is designed to operate in large deployments with potentially millions of edge devices. However, such a deployment is not only prohibitively expensive, but also impractical to ensure the reproducibility of experiments. As such, we resort to a cluster with 68 NVIDIA Tesla P100 GPUs, and emulate up to 1300 participants in each round. Clients are running with PySyft [7] using PyTorch v1.1.0 backend. We emulate the heterogeneous system performance with hundreds of profiles of end-device computational capacity and network bandwidth, and use a large-scale real-world user behavior dataset [75] to emulate the dynamics of client availability over time (Details in Appendix A.2). To mitigate stragglers in each training round, we employ the widely-used mechanism specified in real FL deployments [20], where we collect updates from the first K completed participants out of $1.3K$ participants, and K is 100 by default.

Datasets and models. We run three categories of applications with four real-world datasets of different scales:

- *Speech Recognition*: the small-scale Google speech dataset [72], with 105k speech commands over 3k clients. We train a convolutional neural network model (ResNet-34 [36]) to recognize the command among 35 categories.
- *Image Classification*: the middle-scale OpenImage [3] dataset, with 1.5 million images over 14k clients. We train MobileNet [65] and ShuffleNet [77] models to classify the image among 600 categories.
- *Language Modeling*: the large-scale StackOverflow [9] and Reddit [8] dataset, with 0.3 and 1.6 million clients respectively. We train next word predictions with Albert.

These applications are widely used in real end-device applications [73], and these models are designed to be lightweight.

Parameters. The minibatch size of each participant is 16 in speech recognition, and 32 in other tasks. The initial learning rate for Albert model is 4e-5, and 0.04 for other models. These configurations are consistent with those reported in the literature [34]. In configuring the training selector, Oort uses the popular time-based exploration factor [15], where the initial exploration factor is 0.9, and decreased by a factor 0.98 after each round when it is larger than 0.2. The step window of pacer W is 20 rounds. We set the pacer step Δ in a way that it can cover the duration of next $W \times K$ clients in the descending order of explored clients' duration, and the straggler penalty α to 2. We remove a client from Oort's exploitation list once she has been selected over 10 times.

Metrics. We care about the *time-to-accuracy* performance and *final model accuracy* for model training tasks. For model testing, we measure the *end-to-end* testing duration, which consists of the computation overhead of the solution and the duration of actual computation.

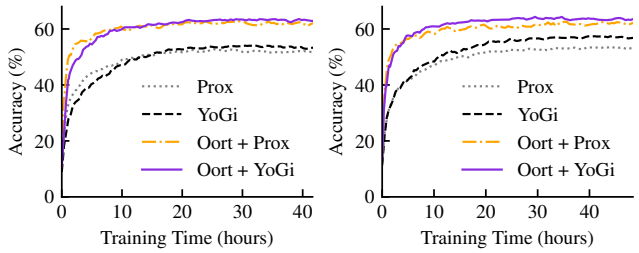
For each experiment, we report the mean value over 5 runs, and error bars show the standard deviation.

7.2 FL Training Evaluation

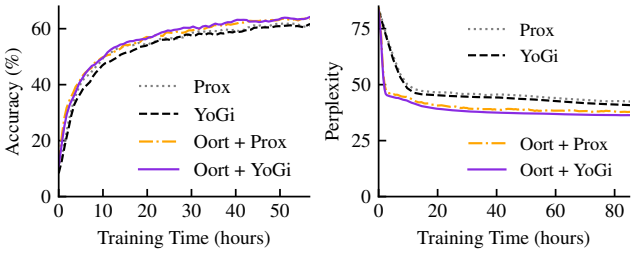
In this section, we evaluate Oort's performance on model training, and employ Prox [49] and YoGi [63]. We refer Prox as Prox running with existing random participant selection, and Prox + Oort is Prox running atop Oort. We use a similar denotation for YoGi. Note that Prox and YoGi optimize the statistical model efficiency for the given participants, while Oort cherry-picks participants to feed them.

7.2.1 End-to-End Performance

Table 1 summarizes the key result of time-to-accuracy performance, while Figure 8 reports the timeline of training to achieve different accuracy. In the rest of the evaluations, we



(a) MobileNet (Image). (b) ShuffleNet (Image).



(c) ResNet (Speech). (d) Albert (LM).

Figure 8: ***Time-to-Accuracy performance. A lower perplexity is better in the language modeling (LM) task.***[10]

report the Albert performance on Reddit dataset for brevity.

Oort improves time-to-accuracy performance. We notice that Oort achieves large speedups to reach the target accuracy (Table 1). Oort reaches the target 3.3×-14.1× faster in terms of wall clock time on the middle-scale OpenImage dataset; speedup on the large-scale Reddit and StackOverflow dataset is 7.3×-9.1×. Understandably, these benefits decrease when the total number of clients is small, as shown on the small-scale Google Speech dataset (1.2×-1.3×).

These time-to-accuracy improvements stem from the comparable benefits in statistical model efficiency and system efficiency (Table 1). Oort takes 1.8×-4.8× fewer training rounds on OpenImage dataset to reach the target accuracy, which is better than that of language modeling tasks (1.3×-2.1×). This is because real-life images often exhibit greater heterogeneity in data characteristics than the language dataset, whereas the large population of language datasets leaves a great potential to prioritize clients with faster system speed.

Oort improves final model accuracy. When the model converges, Oort achieves 6.6%-9.8% higher final accuracy on OpenImage dataset, and 3.1%-4.4% better perplexity on Reddit dataset (Figure 8). Again, this improvement on Google Speech dataset is smaller (1.3% for Prox and 2.2% for YoGi) due to the small scale of clients. These improvements attribute to the exploitation of high statistical utility clients. Specifically, the statistical model accuracy is determined by the quality of global aggregation. Without cherry-picking participants in each round, clients with poor statistical model utility can dilute the quality of aggregation. As such, the model may converge to suboptimal performance. Instead, models running with Oort concentrate more on clients with high statistical utility, thus achieving better final accuracy.

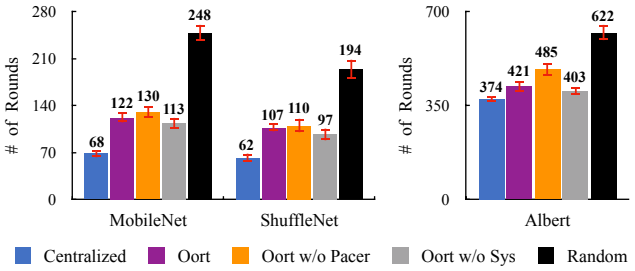


Figure 9: ***Number of rounds to reach the target accuracy.***

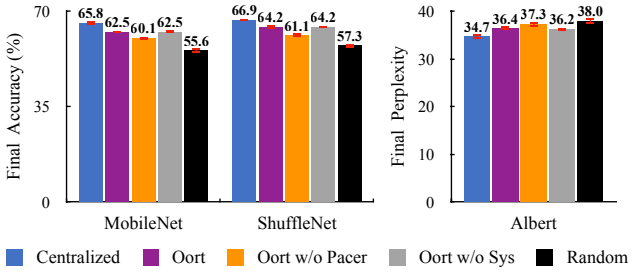


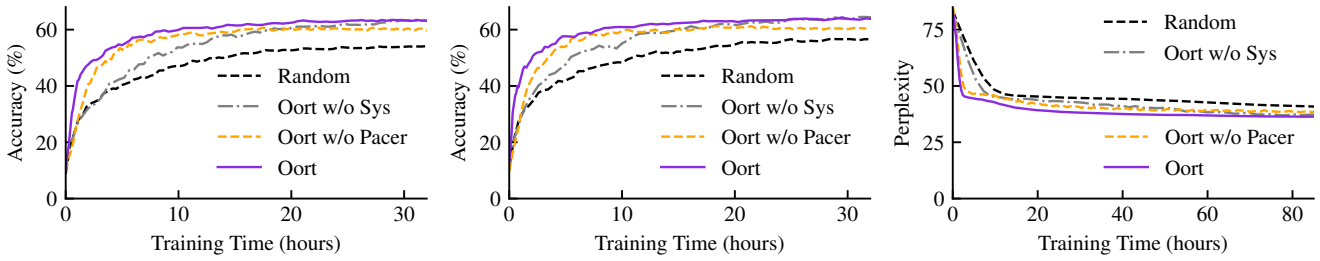
Figure 10: ***Breakdown of final model accuracy.***

7.2.2 Performance Breakdown

We next delve into the improvement on middle- and large-scale datasets, as they are closer to real FL deployments. We break down our knobs designed for striking the balance between statistical and system efficiency: (i) (*Oort w/o Pacer*): We disable the pacer that guides the aggregation efficiency. As such, it keeps suppressing low-speed clients, and the training can be restrained among low-utility but high-speed clients; (ii) (*Oort w/o Sys*): We further totally remove our benefits from system efficiency by setting α to 0, so Oort blindly prioritizes clients with high statistical utility. We take YoGi for analysis, because it outperforms Prox most of the time.

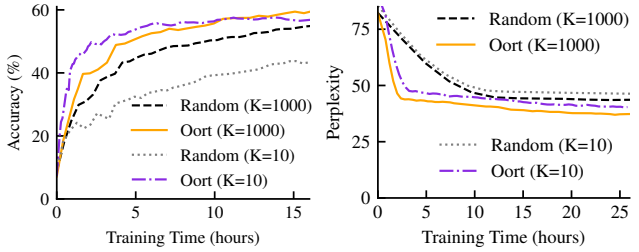
Breakdown of time-to-accuracy efficiency. Figure 11 reports the breakdown of time-to-accuracy performance, where Oort achieves comparable improvement from statistical and system optimizations. Taking Figure 11(b) as example, (i) At the beginning of training, both Oort and (Oort w/o Pacer) improve the model accuracy quickly, because they penalize the utility of stragglers and select clients with higher statistical utility and system efficiency. In contrast, (Oort w/o Sys) only considers the statistical utility, resulting in longer rounds. (ii) As training evolves, the pacer in Oort gradually relaxes the constraints on system efficiency, and admits clients with relatively low speed but higher statistical utility, which ends up with the similar final accuracy of (Oort w/o Sys). However, (Oort w/o Pacer) relies on a fixed system constraint and suppresses valuable clients with high statistical utility but low speed, leading to suboptimal final accuracy.

[10] A low perplexity indicates the probability distribution of word predictions fits the ground truth better, thus implying better accuracy.



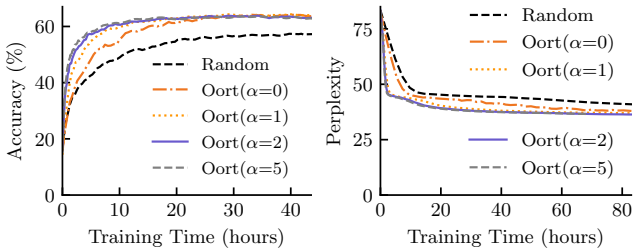
(a) MobileNet (Image). (b) ShuffleNet (Image). (c) Albert (LM).

Figure 11: ***Breakdown of Time-to-Accuracy performance with YoGi, when using different participant selection strategies.***



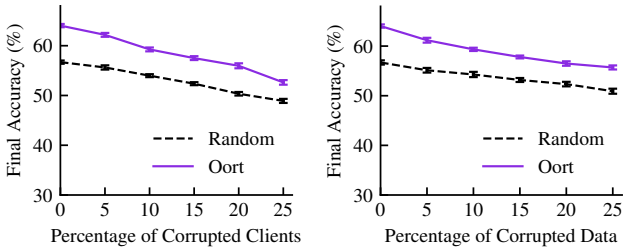
(a) ShuffleNet (Image). (b) Albert (LM).

Figure 12: ***Oort outperforms in different scales of participants.***



(a) ShuffleNet (Image). (b) Albert (LM).

Figure 13: ***Oort improves performance across penalty factors.***



(a) Corrupted clients. (b) Corrupted data.

Figure 14: ***Oort still improves performance under outliers.***

Breakdown of statistical model efficiency. We consider an *upper-bound* statistical model efficiency by creating a centralized case, where all training samples are evenly distributed to K participants. Using the target accuracy in Table 1, Oort can efficiently approach this upper bound by incorporating different components (Figure 9). Oort is within 2× of the upper-bound to achieve the target accuracy, and (Oort w/o Sys) performs the best in statistical model efficiency, because (Oort w/o Sys) always grasps clients with higher statistical utility, whereas it falls short in time-to-accuracy performance because of ignoring the system efficiency. Moreover, by introducing the pacer, Oort achieves 2.4%-3.1% better accuracy than (Oort w/o Pacer), and is merely about 2.7%-3.3% worse than the upper-bound final model accuracy (Figure 10).

7.2.3 Sensitivity Analysis

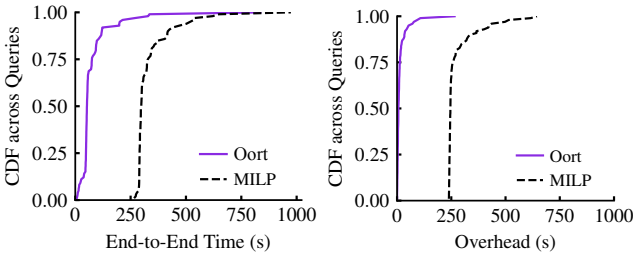
Impact of number of participants K. We evaluate Oort across different scales of participants in each round, where we cut off the training after 200 rounds given the diminishing rewards. We observe that Oort improves time-to-accuracy efficiency across different number of participants (Figure 12), and having more participants in FL indeed receives diminishing rewards. This is because taking more participants (i) is similar to having a large batch size, which is confirmed to be even negative to round-to-accuracy performance [52]; (ii) can lead to longer rounds due to stragglers when the number of clients is limited (e.g., K=1000 on OpenImage dataset).

Impact of penalty factor α on stragglers. Oort uses the penalty factor α to penalize the utility of stragglers in participant selection, whereby it adaptively prioritizes high system efficiency participants. Figure 13 shows that Oort achieves persistent improvements and outperforms its counterparts on different models across different non-zero penalty factors.

Impact of outliers. We investigate the robustness of Oort by introducing outliers manually. Following the popular adversarial ML setting [30], we randomly flip the ground-truth data labels of the OpenImage dataset to any other categories, resulting in artificially high utility. We consider two practical scenarios with the ShuffleNet model: (i) Corrupted clients: labels of all training samples on these clients are flipped (Figure 14(a)); (ii) Corrupted data: each client uniformly flips a subset of her training samples (Figure 14(b)). We notice Oort still outperforms across all degrees of corruption.

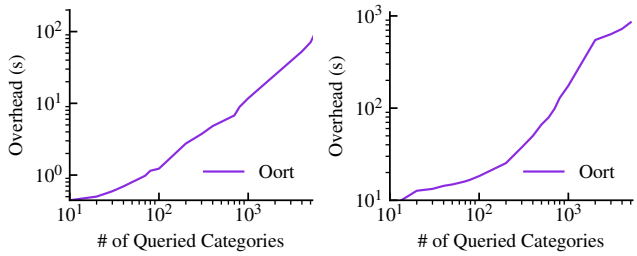
7.3 FL Testing Evaluation

In this section, we evaluate the clairvoyant selector and the non-clairvoyant one for model testing tasks. Our primary goal is to satisfy the developer requirements on data distribution.



(a) OpenImage (Testing duration). (b) OpenImage (Overhead).

Figure 15: ***Oort outperforms MILP in clairvoyant FL testing.***



(a) StackOverflow (0.3M clients). (b) Reddit (1.6M clients).

Figure 16: ***Oort scales to millions of clients, while MILP did not complete on any query.***

7.3.1 Clairvoyant Selector

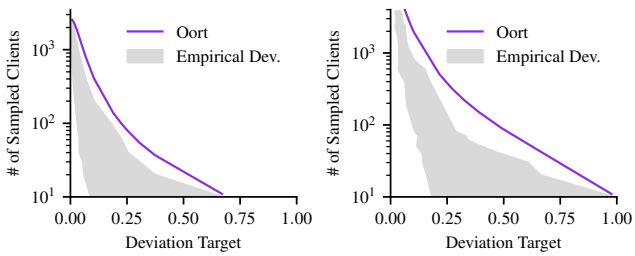
Oort outperforms MILP. We start with the middle-scale OpenImage dataset and compare the end-to-end testing duration of Oort and MILP. In our experiment, we generate 200 queries using the form "Give me X representative samples", where we sweep X from 4k to 200k and budget B from 100 to 5k participants. We simulate the validation time of MobileNet on participants selected by these strategies.

Figure 15(a) shows the end-to-end testing duration. We observe Oort outperforms MILP by 4.7× on average. This is because Oort suffers little computation overhead by greedily reducing the search space of MILP. As shown in Figure 15(b), MILP takes 274 seconds on average to complete the participant selection, while Oort only takes 15 seconds.

Oort is scalable. We further investigate Oort's performance on the large-scale StackOverflow and Reddit dataset with millions of clients, where we take 1% of the global data as the requirement, and sweep the number of interested categories from 1 to 5k. Figure 16 shows even though we gradually magnify the search space of participant selection by introducing more categories, Oort can serve our requirement in a few minutes at the scale of millions of clients, while MILP fails to generate the solution decision for any query.

7.3.2 Non-Clairvoyant Selector

Oort can cap data deviation. Figure 17 reports Oort's performance on serving different deviation targets, with respect to the global distribution. We sweep the number of selected clients from 10 to 4k, and randomly select each given number of participants over 1k times to empirically search their



(a) Google Speech. (b) Reddit.

Figure 17: ***Oort can cap data deviation for all targets. Shadow indicates the empirical [min, max] range of the x-axis values over 1000 runs given the y-axis input.***

possible deviation. We notice that for a given deviation target, (i) different workloads require distinct number of participants. For example, to meet the target of 0.05 divergence, the Speech dataset uses 6× less participants than the Reddit attributing to its smaller heterogeneity (e.g., tighter range of the number of samples); (ii) with the Oort-determined number of participants, no empirical deviation exceeds the target, showing the effectiveness of Oort in satisfying the deviation target; (iii) Oort guides the number of participants needed to guarantee the deviation target, which reduces the cost of expanding participant set arbitrarily.

8 Related Work

Federated Learning Federated learning [44] is a distributed machine learning paradigm in a network of end devices, wherein Prox [49] and YoGi [63] are state-of-the-art optimizations in tackling data heterogeneity. Recent efforts in FL have been focusing on improving communication efficiency [38,55] or compression schemes [14], ensuring privacy by leveraging multi-party computation (MPC) [21] and differential privacy [31], or tackling heterogeneity by reinventing ML algorithms [50,70]. However, they underperform in FL because of the suboptimal participant selection they rely on.

Datacenter Machine Learning Distributed machine learning in datacenters has been well-studied [40,59,61], wherein they assume shared data and relatively homogeneous workers [33,53]. Although developer requirements and models can still be the same, the client heterogeneity makes FL much more challenging. We aim at enabling them in FL. While bearing some resemblance in prioritizing important training samples [43,46,78], we consider both statistical and system efficiency to select participants at scale.

Privacy-preserving Data Analytics To gather sensitive statistics from user devices, several differentially private systems add noise to user inputs locally to ensure privacy [29], whereas some assume a trusted third party, which only adds noise to the aggregated raw inputs [19], or use MPC to enable global differential privacy without a trusted party [64]. Our work is orthogonal to them, but developers can leverage them to collect client information for the clairvoyant model testing.

9 Conclusion

In this paper, we presented Oort to allow guided participant selection for FL developers. Compared to existing participant selection mechanisms, Oort achieves large speedups in time-to-accuracy performance for federated training by picking clients with high data and system utility, and it is able to serve developer requirements on data distribution efficiently during testing even at the scale of millions of clients.

References

[1] AI Benchmark: All About Deep Learning on Smartphones. http://ai-benchmark.com/ranking_deeplearning_detailed.html.

[2] Federated AI Technology Enabler. https://www.fedai.org/.

[3] Google Open Images Dataset. https://storage.googleapis.com/openimages/web/index.html.

[4] Google's Sundar Pichai: Privacy Should Not Be a Luxury Good. https://www.nytimes.com/2019/05/07/opinion/google-sundar-pichai-privacy.html.

[5] Gurobi. https://www.gurobi.com/.

[6] MobiPerf. https://www.measurementlab.net/tests/mobiperf/.

[7] PySyft. https://github.com/OpenMined/PySyft.

[8] Reddit Comment Data. https://files.pushshift.io/reddit/comments/.

[9] Stack Overflow Data. https://cloud.google.com/bigquery/public-data/stackoverflow.

[10] Stanford Puffer. https://puffer.stanford.edu/.

[11] TensorFlow Federated. https://www.tensorflow.org/federated.

[12] Transformers. https://github.com/huggingface/transformers.

[13] Martín Abadi, Andy Chu, Ian Goodfellow, and et al. Deep learning with differential privacy. In *CCS*, 2016.

[14] Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: randomized quantization for communication-optimal stochastic gradient descent. In *arxiv.org/abs/1610.02132*, 2016.

[15] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. In *Machine Learning*, 2002.

[16] Sean Augenstein, H. Brendan McMahan, and et al. Generative models for effective ML on private, decentralized datasets. In *ICLR*, 2020.

[17] Rémi Bardenet and Odalric-Ambrym Maillard. *Concentration inequalities for sampling without replacement*. Bernoulli Society for Mathematical Statistics and Probability, 2015.

[18] Patrice Bertail, Emmanuelle Gautherat, and Hugo Harari-Kermadec. Exponential bounds for multivariate self-normalized sums. *Electron. Commun. Probab.*, 13:no. 57, 628–640, 2008.

[19] Andrea Bittau, Úlfar Erlingsson, and et al. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*, 2017.

[20] Keith Bonawitz, Hubert Eichner, and et al. Towards federated learning at scale: System design. In *MLSys*, 2019.

[21] Keith Bonawitz, Vladimir Ivanov, and et al. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.

[22] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data validation for machine learning. In *MLSys*, 2019.

[23] Mingqing Chen, Ananda Theertha Suresh, Rajiv Mathews, Adeline Wong, Cyril Allauzen, Francoise Beaufays, and Michael Riley. Federated learning of n-gram language models. In *ACL*, 2019.

[24] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, 2017.

[25] Michal Daszykowski, Beata Walczak, and D.L. Massart. Representative subset selection. *Analytica Chimica Acta*, 468:91–103, 09 2002.

[26] Apple Differential Privacy Team. Learning with privacy at scale. In *Apple Machine Learning Journal*, 2017.

[27] S. Dutta, D. Wei, H. Yueksel, P. Y. Chen, S. Liu, and K. R. Varshney. Is there a trade-off between fairness and accuracy? a perspective using mismatched hypothesis testing. In *ICML*, 2020.

[28] Hubert Eichner, Tomer Koren, H. Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In *ICML*, 2019.

[29] Ulfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

[30] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security Symposium*, 2020.

[31] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. In *NeuIPS*, 2017.

[32] Siddharth Gopal. Adaptive sampling for SGD by exploiting side information. In *ICML*, 2016.

[33] Juncheng Gu, Mosharaf Chowdhury, and et al. Tiresias: A GPU cluster manager for distributed deep learning. In *NSDI*, 2019.

[34] Andrew Hard, Kanishka Rao, and et al. Federated learning for mobile keyboard prediction. In *arxiv.org/abs/1811.03604*, 2019.

[35] Florian Hartmann, Sunah Suh, Arkadiusz Komarzewski, Tim D. Smith, and Ilana Segall. Federated learning for ranking browser history suggestions. In *arxiv.org/abs/1911.11807*.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[37] Kevin Hsieh, Ganesh Ananthanarayanan, and et al. Focus: Querying large video datasets with low latency and low cost. In *OSDI*, 2018.

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

[39] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The Non-IID data quagmire of decentralized machine learning. In *arxiv.org/abs/1910.00189*.

[40] Zhihao Jia, Oded Padon, and et al. TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In *SOSP*, 2019.

[41] Junchen Jiang, Rajdeep Das, and et al. Via: Improving internet telephony call quality using predictive relay selection. In *SIGCOMM*, 2016.

[42] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew A. Chien. Networked cameras are the new big data clusters. In *HotEdgeVideo*, 2019.

[43] Tyler B. Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In *NeurIPS*, 2018.

[44] Peter Kairouz, H. Brendan McMahan, and et al. Advances and open problems in federated learning. In *arxiv.org/abs/1912.04977*.

[45] Angelos Katharopoulos and Francois Fleuret. Biased importance sampling for deep neural network training. In *arxiv.org/pdf/1706.00043*.

[46] Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *ICML*, 2018.

[47] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020.

[48] Liam Li, Kevin Jamieson, and et al. A system for massively parallel hyperparameter tuning. In *MLSys*, 2020.

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

[50] Tian Li, Manzil Zaheer, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *ICLR*, 2020.

[51] Wenqi Li, Fausto Milletari, and Daguang Xu. Privacy-preserving federated brain tumour segmentation. In *Machine Learning in Medical Imaging*, 2019.

[52] Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local SGD. In *ICLR*, 2020.

[53] Kshiteej Mahajan, Arjun Balasubramanian, and et al. Themis: Fair and efficient GPU cluster scheduling. In *NSDI*, 2020.

[54] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Suresh. Three approaches for personalization with applications to federated learning. In *arxiv.org/abs/2002.10619*, 2020.

[55] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[56] William Mendenhall, Robert J Beaver, and Barbara M Beaver. *Introduction to probability and statistics*. Cengage Learning, 2012.

[57] William Mendenhall, Robert J Beaver, and Barbara M Beaver. *Introduction to probability and statistics*. Cengage Learning, 2012.

[58] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *ICML*, 2019.

[59] Deepak Narayanan, Aaron Harlap, and et al. Pipedream: Generalized pipeline parallelism for dnn training. In *SOSP*, 2019.

[60] Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. Federated adversarial domain adaptation. In *ICLR*, 2020.

[61] Yanghua Peng, Yibo Zhu, and et al. A generic communication scheduler for distributed dnn training acceleration. In *SOSP*, 2019.

[62] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166, 2015.

[63] Sashank Reddi, Zachary Charles, and et al. Adaptive federated optimization. In *arxiv.org/abs/2003.00295*, 2020.

[64] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *SOSP*, 2019.

[65] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[66] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, and Felix Biessmann. Automating large-scale data quality verification. In *VLDB*, 2018.

[67] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. Understanding and benchmarking the impact of gdpr on database systems. In *VLDB*, 2020.

[68] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. The seven sins of personal-data processing systems under GDPR. In *HotCloud*, 2019.

[69] Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and H. Brendan McMahan. Distributed mean estimation with limited communication. In *ICML*, 2017.

[70] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. In *MLSys*, 2019.

[71] Kangkang Wang, Rajiv Mathews, Chloe Kiddon, Hubert Eichner, Francoise Beaufays, and Daniel Ramage. Federated evaluation of on-device personalization. In *arxiv.org/pdf/1910.10252*.

[72] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. In *arxiv.org/abs/1804.03209*.

[73] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *WWW*, 2019.

[74] Francis Y. Yan, Hudson Ayers, and et al. Learning in situ: a randomized experiment in video streaming. In *NSDI*, 2020.

[75] Chengxu Yang, Qipeng Wang, and et al. Heterogeneity-aware federated learning. In *arxiv.org/pdf/2006.06983*, 2020.

[76] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving Google keyboard query suggestions. In *arxiv.org/abs/1812.02903*.

[77] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[78] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *ICML*, 2015.

A Analysis of Real-world Dataset

Here we provide a detailed description of the datasets used in the paper.

A.1 Dataset of Samples

A summary of statistics for these datasets can be found in Table 2.

Dataset	# of Clients	# of Samples
Google Speech [72]	2,618	105,829
OpenImage [3]	14,477	1,672,231
StackOverflow [9]	342,477	135,818,730
Reddit [8]	1,660,820	351,523,459

Table 2: ***Dataset statistics.***

Google Speech Commands. A speech recognition dataset [72] with over ten thousand clips of one-second-long duration. Each clip contains one of the 35 common words (e.g., digits zero to nine, "Yes", "No", "Up", "Down") spoken by thousands of different people.

OpenImage. OpenImage [3] is a vision dataset collected from Flickr, an image and video hosting service. It contains a total of 16M bounding boxes for 600 object classes (e.g., Microwave oven). We clean up the dataset according to the provided indices of clients. In our evaluation, the size of each image is 96×96.

Reddit and StackOverflow. Reddit [8] (StackOverflow [9]) consists of comments from the Reddit (StackOverflow) website. It has been widely used for language modeling tasks, and we consider each user as a client. In our experiments, we restrict to the 30k most frequently used words, and represent each sentence as a sequence of indices corresponding to these 30k frequently used words. We use Transformers [12] to tokenize these sequences with a block size 64.

In our experiments, we use 90% of the dataset as the training set, and the rest 10% as the held-out testing set.

A.2 Dataset of System Performance and Availability

Heterogeneous system performance. We use the AIBench [1] dataset and MobiPerf [6] dataset. AIBench dataset provides the training time of different models across a wide range of devices. As specified in real FL deployments [20, 34], we focus on the capability of mobile devices that have > 2GB RAM in this paper. To understand the network capacity of these devices, we clean up the MobiPerf dataset, and analyze the available bandwidth when they are connected with WiFi, which is preferred for FL as well [44]. In our evaluations, we configure the system performance of each client via sampling by the distribution of these profiles.

Availability of clients. We use a large-scale real-world user behavior dataset [75]. It comes from a popular input method app (IMA) that can be downloaded from Google Play, and covers 136k users and spans one week from January 31st to February 6th in 2020. This dataset includes 180 million trace items (e.g., battery charge or screen lock) and we consider user devices that are in charging to be available, as specified in real FL deployments [20].

B Proving Benefits of Statistical Utility

We follow the proof of importance sampling to show the advantage of our statistical utility in theory. The convergence speed of Stochastic Gradient Descent (SGD) can be defined as the reduction R of the divergence of model weight $\mathbf{w}$ from its optimal $\mathbf{w}^*$ in two consecutive round t and $t+1$ [46, 78] :

$$R = \mathbb{E}\big[\, \| \mathbf{w_t} - \mathbf{w}^* \|_2^2 - \| \mathbf{w_{t+1}} - \mathbf{w}^* \|_2^2 \,\big] \tag{2}$$

How does oracle sampling help in theory? If the learning rate of SGD is η and we use loss function L to measure the training loss between input features x and the label y, then $\mathbf{w_{t+1}} = \mathbf{w_t} - \eta \nabla L(\mathbf{w_t}(x_i), y_i)$. We now set the gradient $G_t = \nabla L(\mathbf{w_t}(x_i), y_i)$ for brevity, then from Eq. (2):

$$\begin{aligned}
R &= -\mathbb{E}\big[(\mathbf{w_{t+1}} - \mathbf{w}^*)^T(\mathbf{w_{t+1}} - \mathbf{w}^*) - (\mathbf{w_t} - \mathbf{w}^*)^T(\mathbf{w_t} - \mathbf{w}^*)\big] \\
&= -\mathbb{E}\big[\mathbf{w_{t+1}^T w_{t+1}} - 2\mathbf{w_{t+1}}\mathbf{w}^* - \mathbf{w_t^T w_t} + 2\mathbf{w_t}\mathbf{w}^*\big] \\
&= -\mathbb{E}\big[(\mathbf{w_t} - \eta \mathbf{G_t})^T(\mathbf{w_t} - \eta \mathbf{G_t}) + 2\eta G_t^T \mathbf{w}^* - \mathbf{w_t}^T \mathbf{w_t}\big] \\
&= -\mathbb{E}\big[-2\eta(\mathbf{w_t} - \mathbf{w}^*)G_t + \eta^2 G_t^T G_t\big] \\
&= 2\eta(\mathbf{w_t} - \mathbf{w}^*)\mathbb{E}[G_t] - \eta^2 \mathbb{E}[G_t]^T \mathbb{E}[G_t] - \eta^2 Tr(\mathbb{V}[G_t])
\end{aligned} \tag{3}$$

It has been proved that optimizing the first two terms of Eq. (3) is intractable due to their joint dependency on $\mathbb{E}[G_t]$, however, one can gain a speedup over random sampling by intelligently sampling important data bins to minimize $Tr(\mathbb{V}[G_t])$ (i.e., reducing the variance of gradients while respecting the same expectation $\mathbb{E}[G_t]$) [43, 46]. Here, the oracle is to pick bin B_i with a probability proportional to its importance $|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k \in B_i} \| G(k) \|^2}$, where $\| G(k) \|$ is the L2-norm of the unique sample k's gradient $G(k)$ in bin B_i (Please refer to [32] for detailed proof).

How does loss-based approximation help? We have shown the advantage of importance sampling by sampling the larger gradient norm data, and next we present a theoretical proof that motivates our loss-based utility design.

Corollary 0.1. *(Theorem 2 in [45]). Let* $\| G(k) \|$ *denote the gradient norm of any sample k,* $M = \max \| G(k) \|$. *There exists* $K > 0$ *and* $C < M$ *such that* $\frac{1}{K} L(\mathbf{w}(x_k), y_k) + C \geq \| G(k) \|$.

This corollary implies that a bigger loss leads to a large upper bound of the gradient norm. To sample data with a larger

gradient norm, we prefer to pick the one with bigger loss. Moreover, it has been empirically shown that sampling high loss samples exhibits similar variance reducing properties to sampling according to the gradient norm, resulting in better convergence speed compared to naive random sampling [46].

By taking account of the oracle and the effectiveness of loss-based approximation, we propose our loss-based statistical utility design, whereby we achieve the close-to-optimal statistical performance (§7.2.2).

C Determining Size of Participants

We next introduce Lemma 1, which captures how the empirical value of $\bar{X}$ (i.e., average number of samples of participants for category X) deviates from the expectation $E[\bar{X}]$ (i.e., average number of samples of all clients) as the size of participants n varies.

Lemma 1. *For a given tolerance on deviation* ε *and confidence interval* δ *for category* X*, the number of participants* n *we need to achieve* $Pr[|\bar{X} - E[\bar{X}]| < \varepsilon] > \delta$ *requires:*

$$n \geq (N+1) \times \frac{1}{1 - \frac{2N}{\log(1-\delta)} \times (\frac{\varepsilon}{\max\{X\} - \min\{X\}})^2} \quad (4)$$

where N *is the total number of feasible clients, and* $\max\{X\}$ *and* $\min\{X\}$ *denote the maximum and minimum possible number of samples that one client can hold, respectively.*

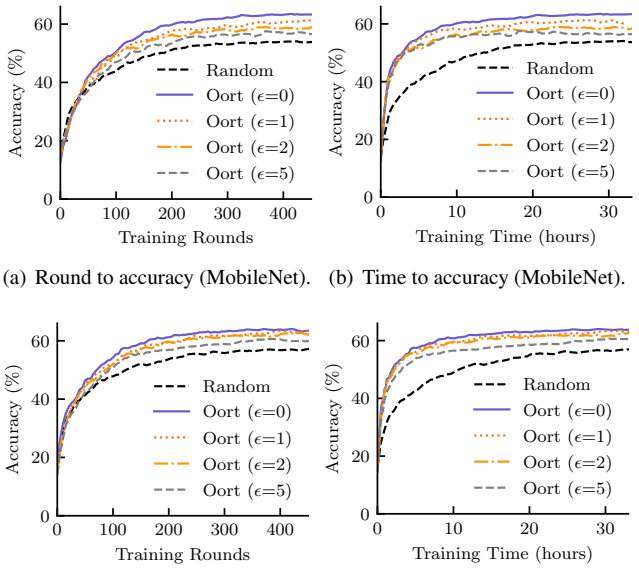
Lemma 1 is a corollary of Hoeffding-Serfling Bound [17], and we omit the detailed proof for brevity. Intuitively, when we have an extremely stringent requirement (i.e., $\varepsilon \to 0$), we have to include more participants (i.e., $n \to N$). When more information of the client data characteristics is available, one can refine this range better. For example, the bound of Eq. (4) can be improved with Chernoff's inequality [17] when the distribution of sample quantities is provided.

Similarly, the multi-category scenario proves to be an instance of multi-variate Hoeffding Bound. Given the developer-specific requirement on each category, the developer may want to figure out how many participants needed to satisfy all these requirements simultaneously (e.g., $Pr[|\bar{X} - E[\bar{X}]| < \varepsilon_x \wedge |\bar{Y} - E[\bar{Y}]| < \varepsilon_y] > \delta$). More discussions are out of the scope of this paper, but readers can refer to [18] for detailed discussions and a complete solution.

D Privacy Concern in Collecting Feedbacks

Depending on different requirements on privacy, we elaborate how Oort respects privacy while outperforming existing mechanisms (§4.3).

Compute aggregated training loss on clients locally. Our statistical utility of a client relies on the aggregated training loss of all samples on that client. The training loss of each sample measures the prediction uncertainties of model on every possible output (e.g., category), and even the one with a correct prediction can generate non-zero training loss [27]. So



(a) Round to accuracy (MobileNet). (b) Time to accuracy (MobileNet).

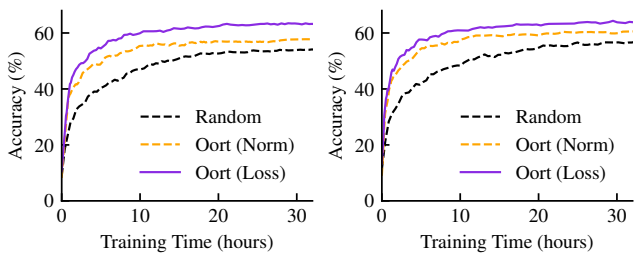
(c) Round to accuracy (ShuffleNet). (d) Time to accuracy (ShuffleNet).

Figure 18: ***Oort improves performance even under noise.***

it does not reveal raw data inputs. Moreover, it does not leak the categorical distribution either, since samples of the same category can own different training losses due to their heterogeneous input features. Note that even when we consider homogeneous input features, a high total loss can be from several high loss samples, or many moderate loss samples.

Add noise to hide the real aggregated loss. Even when clients have very stringent privacy concern on their aggregated loss, clients can add noise to the exact value. Similar to the popular differentially private FL [31], clients can disturb their real aggregated loss by adding Gaussian noise (i.e., noise from the Gaussian distribution). In fact, Oort can tolerate this noisy statistical utility well, owing to its probabilistic selection from the pool of high-utility clients, wherein teasing apart the top-k% utility clients from the rest is the key.

We first prove that Oort is still very likely to select high-utility clients even in the presence of noise. To pick K participants out of N all feasible clients, there are totally $\binom{N}{K}$ possible combinations. We denote these combinations as X_i and sort them $X_1 \leq \ldots \leq X_n$ by the ascending order of total utility. Adding noise to each client ends up with an accumulated noise on X_i. Thereafter, X_i turns to random variables $\mathbf{X}_i$ that follow the distribution of accumulated noise. Specifically, distribution of $\mathbf{X}_i$ is equivalent to shifting the distribution of noise horizontally by a constant X_i. Given that noise added to X_i follows the same distribution, (i) $\mathbf{X}_i$ experiences the same standard deviation for every i; (ii) the expectation of $\mathbf{X}_i$ is the sum of X_i and the expectation of noise. Note that adding a constant (i.e., the expectation of noise here) to the inequality does not change its properties, so we still have $\mathbb{E}[\mathbf{X}_1] \leq \ldots \leq \mathbb{E}[\mathbf{X}_n]$. As such, we are more likely to select high-utility clients (i.e., combination $\mathbf{X}_i$ with higher $\mathbb{E}[\mathbf{X}_i]$)



(a) MobileNet. (b) ShuffleNet.

Figure 19: ***Oort outperforms with different utility definitions.***

in sampling when picking i with the highest value of X_i.

We next show the superior empirical performance of Oort over its counterparts under noise. In this experiment, we add noise from the Gaussian distribution $Gaussian(0, \sigma^2)$, and investigate Oort's performance with different σ. Similar to differential FL [31], we define $\sigma = \varepsilon \times Mean(real\_value)$, where $Mean(real\_value)$ is the average of real value without noise. Note that we take this $real\_value$ as reference for the ease of presentations, and developers can refer to other values. As such, a large ε implies larger variance in noise, thus providing better privacy by disturbing the real value significantly. We report the statistical efficiency after adding noise to the statistical utility (Fig 18(a) and Fig 18(c)), as well as the time-to-accuracy performance (Fig 18(b) and Fig 18(d)). We observe that Oort still improves performance across different amount of noise, and is robust even when the noise is large (e.g., $\varepsilon = 5$ is often considered to be very large noise [13]).

Rely on gradient norm of batches. For case where clients are even reluctant to report the noisy loss, we introduce an alternative statistical utility to drive our exploration-exploitation based client selection. Our intuition is to use the gradient norm of batches to approximate the gradient norm of individual samples $\nabla f(k)$ in the oracle importance $|B_i|\sqrt{\frac{1}{|B_i|}\sum_{k\in B_i} \| \nabla f(k) \|^2}$. In mini-batch SGD, we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t - learning\_rate \times \frac{1}{batch\_size} \times \sum_{k\in batch} \nabla f(k)$$

where w_t is the model weights at time t. Now, we can use the gradient norm of batches (i.e., $\| \mathbf{w}_{t+1} - \mathbf{w}_t \|^2$) to approximate $\| \nabla f(k) \|^2$, and they become equivalent when the batch size is 1. Note that today's FL is already collecting the model updates (i.e., $\mathbf{w}_{t+1} - \mathbf{w}_t$), so we are not introducing additional information. As such, we consider the client with larger accumulated gradient norm of batches to be more important.

We report the empirical performance of this approximation and the loss-based statistical utility using YoGi. As shown in Fig 19, Oort achieves superior performance over the random selection, and the loss-based utility is better than its counterparts. This is because the approximation accuracy with the norm of batches decreases when using mini-batch SGD, whereas mini-batch SGD is more popular than the single-sample batch in ML.