PRIMES (Price Incentive Model Efficiency System): An Incentive-Aligned Agent Selection Strategy for Federated Learning

Jessica Chen[1]
jessica_chen@college.harvard.edu

Jared Ni[1]
jaredni@college.harvard.edu

Gary Wu[1]
garywu@college.harvard.edu

***Abstract*—Federated learning has emerged as a promising approach for training machine learning models across a decentralized network of clients while preserving data privacy. However, one significant challenge in federated learning systems is ensuring clients are motivated to provide high-quality data and training updates. While prior works address efficient participant selection in federated learning, they do not consider the misreporting and free-riding problem, in which clients perform well enough in the FL training process to receive the final model, while minimizing the amount of work they perform or have poor data quality. This paper presents PRIMES, a novel approach to address this challenge by aligning incentives: clients are selected based on next-step loss and paid as a function of, largely, their global performance. Because a client's good performance on the global test data, not selection, guarantees payment, clients are thus incentivized to report truthfully.**

We implement PRIMES on Flower FL with a separate "back-channel" gRPC server. Simulation results show that it achieves higher accuracy per round (1.5-2x) compared to random agent selection and Oort's clipping method, as well as faster times to convergence (10-33%) for client populations with varying degrees of data corruption.

We have made our code publicly available here: https://github.com/jared-ni/PRIMES-cs243-final

I. Introduction

In federated learning, various clients are selected to perform training and provide updates to some aggregator that then iterates over multiple batches of clients in order to improve the end model. Typically, all of the clients involved in this process are rewarded with the final model. Our project takes some inspiration from two areas of focus within the Federated Learning literature: agent selection and incentives.

Our goal is to create an agent selection system that is robust to agent strategizing untruthful reports. We illustrate our use case with the following example.

Suppose we are in the autonomous vehicle development industry. An AI company is trying to train their global model to optimize autonomous vehicle algorithms for traffic patterns in the Southwest U.S. region. They generally already know the target distribution, since traffic patterns in the Southwest U.S. are rather predictable, but they still need regular input of autonomous vehicle data from a number of partner manufacturing companies. These manufacturing companies are the ones who collect the road data; they invest in making good autonomous cars, collect data, and sell that data to the AI company for some amount of payment.

Because we assume that the target distribution is known, we want to select for agents with the lowest individual training loss to achieve faster convergence. However, since agents are rational and capable of strategizing, so it's crucial to incentivize agents to report their data quality truthfully.

In the vanilla model, the AI company randomly selects some of these manufacturers for the training process and pays them a flat fee for selection. However, the company soon realizes random selection leads to some potentially bad datasets being used in their global model training. As an attempt to fix this, the AI company tries to use an Oort-like mechanism, but as we describe in further detail in Section II, Oort's clipping mechanism is too heavy-handed in its attempt to remove strategizing agents.

As a result, the AI company uses our strategy, which focuses on the incentive alignment aspect of the agent selection process.

II. Literature Review

Previous work in this field explores the agent selection problem in Federated Learning, the notion that choosing random clients for each training round is a naive strategy, and that one can achieve better performance by considering which clients can contribute best to the model in the next round.

First, we have **Oort**, an improvement upon random agent selection within federated learning systems by examining the amount of training loss generated by each client in previous rounds to select clients for the next round of training. Oort's overall goal is to propose a system to provide further incentives for clients to provide the best data and training possible for a federated learning system [4]. In these systems, there is a well-documented issue of the "free-riding" problem in these systems where clients may perform well enough to be included in federated learning systems to receive the final model, but may do less work or have poor data quality [4]. This means that Oort fails to incentivize clients to do their best, resulting in the free-riding problem, which becomes particularly costly as federated learning systems scale up to support thousands and even millions of clients.

Furthermore, Oort selects clients based on what they claim their data accuracy to be. However, clients may choose to lie and report great data quality in hopes of selection and payment. To mitigate this misreporting, Oort implements the *clip-*

ping strategy, which removes any client that too consistently provides "good" data. Still, clipping is not always effective; the AI company has since realized that by removing clients reporting consistently well-paying data, they are potentially blacklisting good data candidates and incurring false negatives. Overall, Oort's selection method is not strategyproof.

To explore incentive alignment in federated learning, the paper **Mechanisms that Incentivize Data Sharing in Federated Learning** outlines a different paradigm for incentivizing client performance. They propose that clients receive a proportional final model corresponding to the impact that particular client had on the global training model. Since clients can't receive the fully performance model by free-riding, they are then incentivized to provide their best data and compute in order to receive the best model possible at the end of the training rounds. However, there is no reliable way to trim a model to a fine degree of accuracy, and such implementation is not explored in the paper. As such, we can't currently compare our solution to this paper.

To summarize, the status quo for Federated Learning systems gives us three general options: 1. Vanilla FL, 2. Oort's Clipping Method.

1) In Vanilla systems, the server randomly selects some agents for the training process - there is most likely no payment involved. The problem $\rightarrow$ potentially bad datasets.
2) In Oort, we select clients based on claims of agent accuracy. If a client reports great data quality too many times, then we assume the client is lying, then "clip" this client. The problem $\rightarrow$ we potentially remove good agents, and don't incentivize the best performance every time (if agents know that strong performance every time will result in not being included, they may not give their best every round).

Thus, there exists an opportunity to develop a system that can incentivize agents to provide their best data and compute without having to find a method to trim a final model, and overcome the "free-riding" problem while minimizing misreporting, and achieving a lower Time-To-Convergence.

III. Solution Formulation

We now present PRIMES, a strategy-proof price incentive mechanism to address client laziness and the free-riding problem. We inherit a few elements of the previous literature.

The PRIMES service selects the agents with the lowest individual next step loss (NSL). Since we are assuming our global model already knows the target distribution, agents whose local data yields lowest loss when used for the global model should have the data that most aligns with the global distribution. In other words, we can assume these agents have higher data quality that can lead to faster convergence. The aggregator then pays data suppliers according to their accuracy, or how closely their data aligns with the global model distribution.

We present the following algorithm, implemented using the Flower Federated Learning Framework, to perform agent selection. Notably, the agent selection happens after clients self-report their NSL, however the payment is only completed after the training round is completed and the agent's new weights can be compared against the global distribution.

Algorithm 1: Algorithm for Incentive Alignment

```
Input: K, n, num-rounds
Output: Model
/* For first n rounds, choose random
   clients to update weights          */
1  for i in range(n) do
2      train model on random k clients for 1 iteration
3      update global model with aggregate weights
4  for j in range(num-rounds) do
5      send current global model to all clients
6      for all agents do
7          perform 1 forward-pass to get NSL
8          report NSL to PRIMES service
9      PRIMES sort NSL by asc, select top K clients
10     PRIMES communicate agent selection to Server
11     Server trains and aggregates with selected clients
12     Pay selected clients based on weighted average of
         NSL and Loss wrt private global testset
13     Model.update
14 Return Model
```

To show that our algorithm is strategy-proof, we analyze what happens when an agent wishes to mis-report (by reporting an artificially low NSL), and whether the outcomes are favorable for them. Consider two cases: 1. agent misreports with bad data, 2. agent misreports with good data.

In the case where the agent misreports with bad data quality, they have a higher chance of being selected for training, however this doesn't guarantee that they will receive payment, since the payment function is a weighted average of both the NSL report but also their new weights compared against a private global test dataset. In the case that the agent's data quality is bad, then they risk wasting compute resources during training, and additionally cannot guarantee a worthwhile payment that offsets the time and compute cost associated.

Our algorithm also mitigates the "free-riding" problem: in traditional FL systems, once an agent is included into the general population of potential training clients, they will be guaranteed to receive the final model at the end of the training rounds. As stated earlier, our system proposes that clients may not actually care about receiving the model, and instead care more about the payment they receive from lending out their compute power. As a result, their utility received by participating in this system is directly tied to their performance and contribution to the global model improvements. As such, by putting in minimal effort into the system, they can no longer gain the same utility.

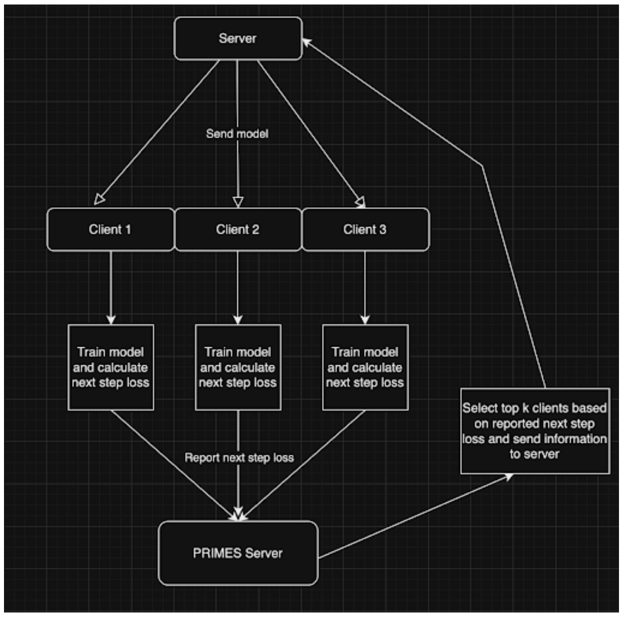
Server
Send model
Client 1
Client 2
Client 3
Train model and calculate next step loss
Train model and calculate next step loss
Train model and calculate next step loss
Report next step loss
Select top k clients based on reported next step loss and send information to server
PRIMES Server

Fig. 1. Algorithm Design

IV. System Design

A. Flower Federated Learning System

The Flower federated learning framework consists of 5 steps in the training process after initializing the global model: model distribution to client nodes, model training using local client data, returning model updates to the central server, aggregating model updates into a new model, and repeating training process until convergence [1]. For performing Federated Learning experiments, Flower, in general, abstracts the difficulty of working with heterogeneous client systems and the difficulty of scaling FL workloads. For our experiment, Flower's flexible and comprehensive nature makes it the ideal FL training framework for incorporating Oort and additional client selection algorithms.

Flower's server-side includes three key components: *ClientManager* maintains the *ClientProxy* objects to connect each client to the server; The FL loop containerizes the entire FL training process; *Strategy* configures the next training round, performs client selection, configuration, parameter update aggregation, and model evaluation. On the client side, clients wait for server messages; upon receiving a server message, clients train models by calling user-provided training and evaluation [1].

B. PRIMES Server & gRPC Design

Our system requires additional information (such as payments, next-step loss, clipped clients) to be communicated between server and client throughout the training process. Incorporating these communications through the Flower framework would be cumbersome and misaligned with the Flower design. Instead, we created a separate PRIMES server with gRPC "back-channels" to facilitate these additional exchanges of information.

For implementing Oort's clipping mechanism, the gRPC server keeps track of the history of selected clients across all rounds, as well as the clients that are "banned" for having been selected too many times consecutively. For implementing the PRIMES strategy, the gRPC server keeps track of clients' next step loss and their loss against the server's test dataset. Using this information, the server can then inform the Flower Strategy which clients should be selected.

While some of these mechanisms could have been implemented through the *ClientManager*, it would have been difficult and misaligned with the ClientManager design to collect next step loss and loss against the local test data through this class. Furthermore, when considering future augmentations of our system, which may expand beyond just client payments to consider equilibrium pricing or server budgets, this separate PRIMES server is better equipped to handle these augmentations.

At the moment, this PRIMES server is most prominently used in the various Strategy classes to select clients for fitting.

C. Strategy Definitions

Within Flower, Oort can be integrated through the *Strategy* abstraction, which is in charge of client selection within Flower. Likewise, our additional client selection can be integrated as the intermediate node during each training round, bridging the communication between Oort and Flower with an added utility price model in the client selection process.

We have implemented three Strategy classes: *FedAvgStrategy*, *ClippingStrategy*, and *PrimesStrategy*. *FedAvgStrategy* is the baseline FL model that randomly selects agents each round via sampling through the *ClientManager*. *ClippingStrategy* calls upon the Primes server during `configure_fit` to sample from non-banned clients.

PrimesStrategy interacts with more elements of the system. In order to make sure the global model is passed to all clients to calculate the next step loss, we modify the `configure_evaluate` function to sample all available clients for evaluation. Each client's `evaluate` method is modified to also calculate the next step loss. All agents' next step losses are then passed to the `aggregate_evaluate` function of *PrimesStrategy*, which then passes it on to the PRIMES server. The *configure_fit* method can then access the PRIMES server for the list of agents selected for best next step loss values. This strategy design leverages the existing parameter-passing framework of Flower client-server interactions to calculate next-step loss and server test-set loss, our two values of interest.

D. Datasets and Corruption

To simulate a real-world scenario with bad actors (clients who provide highly corrupt data to the Federated Learning system), we simulate data corruption by applying a probabilistic random erasing, where for each image tensor in the client's data set, full-image zero out is applied to an image tensor given a probability $\in [0, 1]$. After applying, every pixel of the applied image tensor transforms to a value of

0.0, effectively simulating corrupted data within a data set, where the corrupted data contributes no or negative utility in helping a global model converge in a federated learning setting. A higher corruption probability in a client means a higher corruption proportion in the client's data set, thus we simulate lower-quality clients with a high level of corruption probability and higher-quality clients with a lower level of corruption probability.

V. Evaluation Results

A. Experimental Setup.

Experiments are run locally on MacBook Pro M1 with 16 GB memory with no additional resource restrictions or time limits, and 5 trials were run for each data point. Due to our focus on the number of training rounds (end accuracy or rounds until convergence) rather than training time, and the lack of reliance on hardware components, we found this setting to be sufficient in establishing our experimental basis.

We ran experiments comparing random agent selection, Oort clipping, and PRIMES for agent selection. We will first aim to show that this incentivization method improves training loss over time. Through this process, we model exactly how agents of various quality impact training performance, which we are able to specifically tune using our data corruption implementation described above. We implemented simple neural networks on both the MNIST dataset as well as CIFAR10. This can be done in multiple ways, including by zeroing out a certain number of matrix values randomly, or other methods that we will experiment with. Additionally, we will observe the effects of the free-riding problem. For example, we can artificially substantiate agents that have exceptionally poor datasets but would usually get the same benefit as others with quality data in other FL system practices. We would then compare those results with the ones we see in our model.

B. PRIMES strategy achieves increasingly higher accuracy per training round.

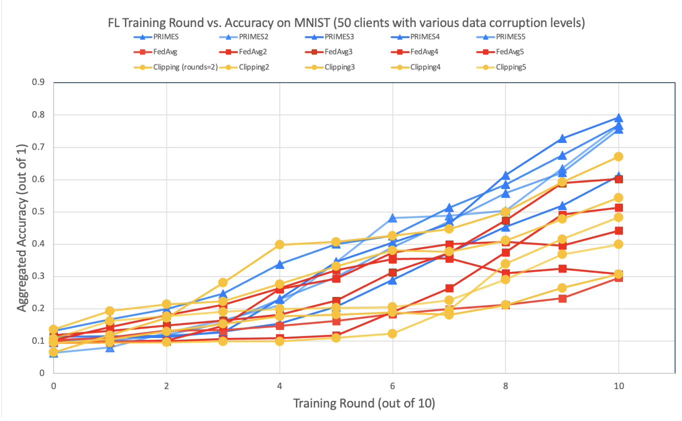

Fig. 2. Accuracy per Training Round for FedAvg, Clipping, and PRIMES

In Figure 2, we compare the three potential agent selection methods (PRIMES, Clipping, FedAvg) with 50 clients and a proportional data corruption split of 0.9, 0.7, 0.5, 0.3, and 0.1, measuring accuracy at each training round. We run each of the three methods 5 times in order to get a sense for variance within each training run.

Our findings are that PRIMES outperforms Clipping by a factor of ∼1.5-2x, and also outperforms FedAvg by ∼2x, meaning that it achieves almost twice the accuracy in the same number of training rounds.

We believe this result is strong evidence that using NSL as a proxy for understanding an agent's potential contribution for the next round is a strong metric. This is obviously better than random agent selection as seen in the graph, but it is interesting to compare PRIMES and Oort's Clipping method in this sense. While both are trying to aim for various goals in incentives, PRIMES encourages constant improvement while Clipping disincentivizes it. Not only does this seem to allow PRIMES to address potential mis-reporting, it also results in better final accuracy after a given number of rounds.

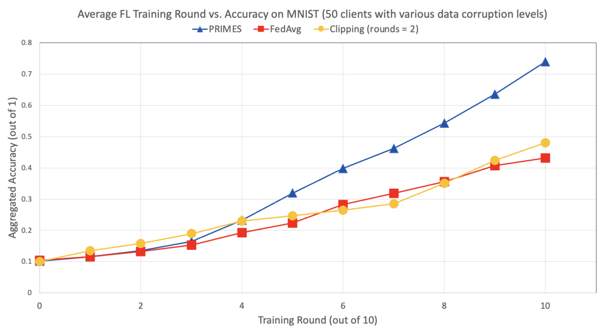

Fig. 3. Averaged Accuracy per Training Round for FedAvg, Clipping, and PRIMES

Additionally, we can look at the averaged accuracy per training round between PRIMES, FedAvg, and Clipping: we notice that the Oort clipping strategy along with FedAvg seem to perform similarly for this distribution of data corruption and client count of 50. Interestingly, the graph starts out relatively equal but then begins to diverge around training round 4.

We believe this is largely due to the way we implemented our PRIMES algorithm, in that the first 3 rounds of training are with randomized agents, meant to provide substantial variability and diversity to the data and weight sets, before then honing in on clients with strong NSL, given that the global weights are then more representative of the entire data distribution.

Further examination of this result could compare how the distribution of data corruption can affect this result. One key intuition our team realized is that high data quality variability among clients is a very suitable environment to deploy our model, since it can prioritize the impact of strong agents, while mitigating the misleading and poor quality data of the worse agents. At the same time, low data quality variability among clients is a sign that regardless of your agent selection algorithm, you are bound to achieve similar results, given the data homogeneity.

Lastly, it is interesting to consider how the number of clients chosen to be selected at each given round can affect performance, which is something a future work can explore.

C. PRIMES achieves convergence faster, especially for populations of more bad clients.

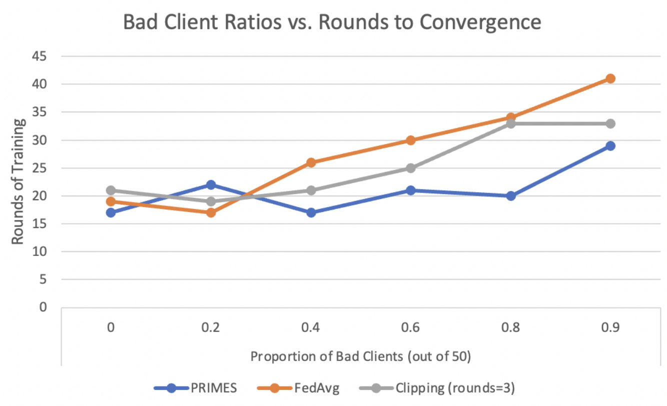

Fig. 4. Rounds to Convergence vs. Proportion of Bad Clients

In Figure 4, we define convergence to be achieved on accuracies $\geq 90\%$. We wanted to see how the different selection strategies would perform in terms of the time to convergence. Here, we define *bad clients* to be those with corruption probabilities of 0.6 and *good clients* to be those with corruption probabilities of 0.1.

We find that, for populations with higher proportions of bad clients, PRIMES performs better than both the vanilla FedAvg and Oort clipping strategies. As expected, lower proportions of bad clients (proportion=0.2) do not see the benefits of the PRIMES server.

D. PRIMES also achieves greater accuracy on more complex datasets.

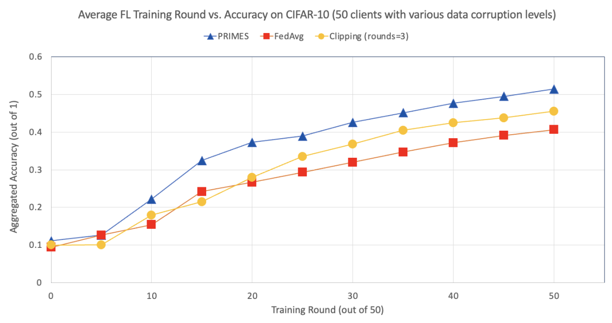

Fig. 5. PRIMES Accuracy per Round on the CIFAR-10 Dataset

Most of the above work was training on the MNIST dataset. Here, we want to see how PRIMES would perform for a more complex dataset such as CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. The images represent a variety of objects and animals.

We find that PRIMES still performs better than the randomly selected FedAvg strategy. Although the convergence rate is not optimal (we only reach 0.5 accuracy after 50 rounds), it is indeed faster than the benchmark. By Round 50, we see that PRIMES has achieved a steady margin of 20% improvement on accuracy from the FedAvg strategy.

Interestingly, the accuracy per round curve in Figure 5 is shaped very differently from the one in Figure 2. Instead of an increasing convex curve, PRIMES yields an approximately increasing concave curve. Further work can be done to further explore how PRIMES can be adapted to different datasets.

VI. Assumptions & Limitations

We make a few large assumptions that make sense for the problem scope laid out in Section I, but are otherwise limiting in practical applications.

1) *Assuming we know the target distribution.* We consider the agents with the lowest loss to be the strategy for fastest convergence. This is opposite to what typical intuition implies; usually, one would select agents with the highest loss so as to expose itself to the most different distribution and learn the fastest. However, by assuming our global model knows the target distribution, it simply needs to select based on agents that align well with this target distribution.
2) *Clipping based on repeated low training loss.* Oort clips based on repeated high loss reports because, in theory, high loss means faster learning. Furthermore, it is infeasible for an agent to continue to provide unique and never-before-seen data on multiple rounds in a row. Therefore, Oort takes repeated selection to be a sign of strategizing. In our system setup, however, we are selecting based on lowest loss. Therefore, the clipping method is banning agents that repeatedly give low next step loss, which may be possible (or even quite likely) to falsely ban an agent that has simply sourced a good dataset.
3) *Defining a robust payment function.* The system payment is defined as some function of both the next step loss and the server test dataset loss values, with more weight placed on the latter. We assume that this payment function is present and drives client decisions accordingly. However, this function does not currently incorporate theoretical considerations for 1) compensating other participating clients for their loss and 2) the budget of either the server or the clients.
4) *Addressing cost occurred for participation and investment by non-selected clients.* While clients are generally incentivized to perform well and report truthfully for selection and payment, the system does not consider the well-meaning clients that continue to incur costs for effort but fail to get selected. The scope of this paper does not tackle the clients' consequences for best-effort investing without payment, nor the server's consequences for potentially driving away otherwise good clients.

VII. Discussion and Future Work

Our simulations found that our proposed PRIMES agent selection strategy outperforms the randomized FedAvg and Oort's clipping strategies by various metrics. When considering rate of accuracy increase per round, we see that by Round 10, PRIMES has achieved approximately 60% improvement over both FedAvg and clipping. When performance against higher proportions of bad clients, we see PRIMES achieving faster time-to-convergence (10%-33%) than either of its counterparts. We also see similar improvements in performance on more complex datasets such as CIFAR-10 (25% improvement over FedAvg).

We attribute this success in training accuracy and speed to the effectiveness of using the next step loss reporting method. We had originally considered other alternatives, such as having clients report the cost they expect to incur in data investments, or clients reporting a certain data quality coefficient. However, neither of these options are particularly helpful indicators in predicting training loss. Next step loss is not only useful, but also computationally cheap to use–clients would only have to run one forward pass using the global model and their own local data.

Since this is a systems focused course, we are mainly interested in analyzing metrics such as time-to-convergence. However, given that our algorithm also aims to solve problems like "free-riding" and mis-reporting, we think there are a few areas to explore further.

1) *Proving Strategyproofness*: It would be interesting to provide a formal proof that our algorithm is strategy-proof, which would include modeling client and server utility, and then incorporating a formal analysis of our PRIMES algorithm.
2) *Formal Analysis of Mis-reporting*: As mentioned before, incentives within FL systems are difficult, since we must operate off of reports (no true information about the client's computation or data), and thus this opens up opportunity for clients to take advantage and lie about their true results. Again, to formally prove that our system disincentivizes mis-reporting, we would need to mathematically model client and server utility, and consider various cases with differing data quality.
3) *Revenue and Pricing Analysis*: Assuming that this system can be used in practice one day, it would be a strong empirical result to show that systems like these can be more cost-efficient than using alternatives like Digital Ocean or AWS. It may also be more realistic to incorporate budgets and corporate credit for later use.
4) *Expanding the system: Marketplace*: While we illustrate our model for one server and multiple clients, there is potential for our proposal to be integrated into a platform or marketplace, connecting compute suppliers with compute requesters around the globe. The systems-level impact for this could be explored further.

VIII. Conclusion

Overall, our PRIMES model successfully aligned incentives between the server and training clients. By using next stop loss as an indicator for selection and global dataset loss on individual data as the determinator of price, PRIMES successfully removes the ability and incentive to falsely report data quality. Empirical tests on this PRIMES system (implemented using FlowerFL and a separate gRPC server) shows that it achieves faster time-to-convergence and higher accuracy per round, especially in the face of populations with higher proportions of bad clients.

IX. Ethical Considerations

Given that Federated Learning is a relatively new paradigm, it is important to consider how advancements in the space must adhere to ethical considerations. From the paper **Advances and Open Problems in Federated Learning**, there are a few relevant points to touch on [4].

There is an issue of model diversity. It is obvious that under our model, if there is a large number of poor-performing agents, and a few high quality agents, they are likely to be selected over and over again, leading to much of the final model being the result of a few clients' datasets. This has large implications for generalized performance but also diversity and robustness. We hope that tuning the right number of randomized client selection rounds at the beginning of training can help to maintain model diversity, along with more advanced techniques.

Additionally, there must be an emphasis on protecting user data. In our system specifically, client data is a treasure, and the PRIMES algorithm is essentially trying to proxy for a client's data quality in each selection round. Such a system implemented in practice must hold data privacy to a high standard.

X. Acknowledgements

We would like to express deep gratitude towards Mark Ting, our wonder Teaching Fellow who has spent probably aggregate a dozen hours on call with us over the past two months, engaged in endless discussion of where to take our project. He has single-handedly seen our project in all its strange, half-shapen forms, and is a big reason why we are able to present this research question in its more-shapen form today.

We would also like to say thank you to Professor Minlan Yu for her sharp feedback, continued support, and infectious enthusiasm. The three of us started this class knowing relatively little about the topic at hand, and walked away with not just exposure to the seminal works of the decade, but also the skills to critically think about such works and where we may contribute. It was fun reflecting on how little we knew about Oort back when you first introduced it to us at the first meeting, and how much farther along we've come.

References

[1] Beutel, Daniel J., et al. ”Flower: A friendly federated learning framework.” (2022).

[2] Lai, Fan, et al. ”Oort: Efficient federated learning via guided participant selection.” 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). 2021.

[3] Karimireddy, Sai Praneeth, Wenshuo Guo, and Michael I. Jordan. ”Mechanisms that incentivize data sharing in federated learning.” arXiv preprint arXiv:2207.04557 (2022).

[4] Kairouz, Peter, et al. ”Advances and open problems in federated learning.” Foundations and Trends® in Machine Learning 14.1–2 (2021): 1-210.