Heterogeneity Aware Federated Learning

Major Qualifying Project

Advisor:

TIAN GUO

Written By:

JUSTIN AQUILANTE
ANN JICHA

A Major Qualifying Project
WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science.

01 SEPTEMBER 2020 - 06 APRIL 2021

Acknowledgements Team FL would like to thank our advisor Professor Tian Guo and our graduate student mentor, Sam Ogden for guiding this project and showing tremendous support throughout the course of the project. Their dedication to providing constructive feedback, encouragement, and their own time at any hour of the day ultimately allowed us to complete the goals of this project despite the challenges we faced along the way. This project absolutely would not have been possible without their support.

Abstract

Machine learning, and more specifically federated learning, is experiencing exponential growth into a variety of industries. Federated learning, training machine learning models on individual user data and aggregating the models, is an increasingly important field due to its current applications rapidly expanding potential, and focus on user privacy. Federated learning provides the convenience of machine learning with additional privacy built in [1]. As more privacy-focused industries look to machine learning for efficient solutions, federated learning is increasingly relevant. Our goal is to evaluate federated learning algorithms and understand how they perform under various scenarios, such as image recognition. In this Major Qualifying Project (MQP), we compare two federated learning algorithms, FedAvg and Fedprox. These algorithms allow models to be aggregated in the federated learning process and can can be customized to user scenarios. To compare both FL algorithms, we tested both algorithms on two datasets: FEMNIST and CELEBA. In order to accomplish our analyses, we implemented a means of preprocessing the datasets we used into a form (HDF5 files) that allows them to be easily utilized in the environments we have prepared. We then prepared an easily replicable testing pipeline for federated learning algorithms. This pipeline included common federated learning analysis tools Google Collaboratory and Tensorflow Federated. We show that under various circumstances, FedProx outperforms FedAvg in *identically and independently distributed* (IID) scenarios. In *non-identically and independently distributed* (NIID) scenarios, the relative performance of the algorithms is less predictable. For example, we found that FedAvg converges 6% faster than FedProx on CELEBA with no stragglers, while there was less than 1% difference between the two on IID. We also tested the impact of adding under performing or "straggling" devices; a common problem in real-world federated learning applications. The extra challenge created by simulating stragglers negatively impacted FedAvg's performance while leaving FedProx mostly unaffected in the MNIST dataset. When the same challenge is introduced with the CELEBA dataset, FedAvg was not able to perform at the same or better level than FedProx; FedProx converged in 50% of the rounds it took FedAvg to converge. However, NIID data again proved unpredictable with FedAvg performing 23% better than FedProx. Our analysis provides an evaluative view on the performance of common federated learning algorithms as well as a means to test other algorithms by expanding testing resources available for comparing them.

Table of Contents

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

Federated learning, which is training machine learning models near to individual user data, is an increasingly important field due to its current applications and rapidly expanding potential.

Machine learning in general, federated learning specifically, is experiencing exponential growth into a variety of industries [2]. Federated learning provides the convenience of machine learning with additional privacy built in [1]. A number of large technology companies have adopted this approach, including the federated learning pioneer Google [3], and industry peer Apple, with its virtual assistant technology [4]. In other fields, medical researchers utilize federated learning to analyze sensitive medical data in order to diagnose and research ailments from cancer to COVID-19 [5] while maintaining patient confidentiality. As industries look to more privacy-minded machine learning for efficient solutions, federated learning is increasingly relevant.

Our goal is to evaluate federated learning algorithms and understand how they perform under various scenarios, such as image recognition. Industry adopters and researchers alike require evaluative resources about federated learning in order to make informed decisions about which algorithm will best fit the requirements of their projects, their users, and their stakeholders. Federated learning is a field in its nascent stages. Federated learning is an approach to machine learning providing the much-needed feature of user privacy, but that built-in privacy will continue to be improved in order to benefit those that utilize this powerful tool [6]. We will provide overall summaries of a selection of existing frameworks (for example, Tensorflow Federated), and testing of common federated learning algorithms in one of these frameworks.

This paper explores two federated learning algorithms: FedAvg and FedProx. In this paper, we show that under the circumstances we test, FedProx outperforms FedAvg in both independent and identically distributed and non independent and identically distributed scenarios, with and without stragglers. To prove this, we tested both algorithms on two datasets: FEMNIST and

CELEBA. FEMNIST is a handwriting dataset intended for benchmarking federated learning algorithms for image classification, and CELEBA is a celebrity face dataset also intended for federated learning benchmarking. We selected these datasets because of their wide use to evaluate federated learning datasets as well as their broad availability. Our survey of these two federated algorithms demonstrates that FedProx handles data that is non identically and independently distributed much better than FedAvg can. We find that FedProx converges to a higher accuracy in less training rounds with both datasets. Additionally, the extra challenge we created by simulating stragglers negatively impacted FedAvg's performance while leaving FedProx mostly unaffected.

This paper builds on an existing body of work evaluating federated learning frameworks. Federated learning is a growing area of research, which is being rapidly adopted in industry. Therefore, a variety of research groups have worked to provide broad evaluations of the field [6], of which we evaluate a few. There are still many opportunities in this area to expand federated learning research. Papers evaluating specific algorithms related to federated learning in depth are often either industry specific, or focus only on comparing a benchmark in order to motivate a proposed new algorithm. Our evaluation focuses on the most broadly applicable federated learning frameworks with the goal of providing a end-to-end comparison of the best options for individuals and corporate entities entering this growing field. We also provide a pipeline for testing more algorithms and datasets in the form of tools that allow for processing datasets into HDF5 files, which can then be loaded into the TensorFlow framework along with an algorithm implementation for evaluation.

The remainder of the paper is structured as follows. The first section 2, is a background section subdivided in order to describe key concepts related to federated learning. Section 2, Methodology 5.1 details our experiments. The third, Results 6, describes our findings. The fourth is our conclusions, a discussion of how this applies to the field of federated learning as a whole, and possibilities for furthering our work. The final section is a Related Works section describing the key papers within this field in order to understand the variety of federated learning tools available.

CHAPTER 2

BACKGROUND

2.1 What Is Federated Learning?

Federated learning is a type of deep machine learning that focuses on sending copies of a central model to individual devices, as opposed to traditional machine learning's approach of sending data [3].

The key attribute of federated learning is that the data used to train models is not centralized, but instead is distributed in a "loose federation" of devices.This is illustrated in Figure 2.2. As opposed to other approaches to training deep learning models where user data is pooled and processed at a centralized location as shown in 2.1, the role of the central server in federated learning is to aggregate training done on participating devices with their own local data. To accomplish this, the model is sent in rounds where devices train the model using a local method such as Stochastic Gradient Descent, send the model back to the server, and then the server aggregates the models together. Then, the model can be sent out again to users for more training [1].

Federated learning offers clear benefits in terms of privacy. This system addresses some privacy concerns of previous machine learning approaches by allowing data to remain decentralized, and therefore harder for attackers to obtain from any user. This is because the data never actually leaves the devices, and attackers must instead bypass device security rather than attempting to obtain data while it is more vulnerable in transit [7]. Despite this advantage, there are a myriad of challenges and possible improvements within federated learning, which we discuss further in Chapter 3.

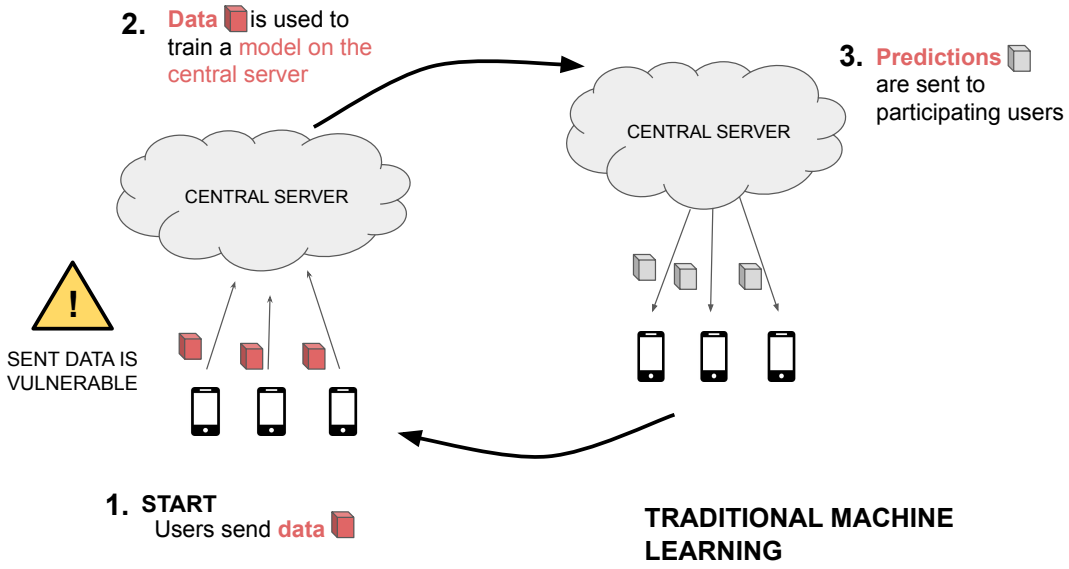


Figure 2.1: *Overview of Machine Learning*

2.1.1 Basic Federated Learning Algorithms

Throughout this paper, we will describe the strengths and weaknesses of a selection of federated learning algorithms (as displayed through the tests we have performed) and how each algorithm could be improved in terms of the key considerations we have selected. This section discusses how federated learning has evolved since it was introduced, from the fairly basic FedSGD to more advanced algorithms like FedProx. We use this discussion as grounds for determining which algorithms to evaluate.

SGD is a fundamental algorithm for machine learning problems which minimizes a loss function to train a model. SGD on its own is not sufficient to solve federated learning efficiently. These issues stem primarily from the many rounds of training required by SGD [3]. We first discuss a number of federated alternatives to naive SGD.

FedSGD. FedSGD is a federated approach to SGD. It uses a large-batch synchronous approach to multi-client learning, which performs better than naive asynchronous SGD training [3]. Each client trains one step on the data, and then sends the updated weights to the server where it averages the weights, and sends the updated model back to the clients for more training. Therefore, there is a significant degree of network activity at each training step.

FedAvg. FedAvg, short for Federated Averaging, builds upon FedSGD. It was introduced by Google Researchers McMahan *et al.*in 2016 [3], and remains a common baseline algorithm for

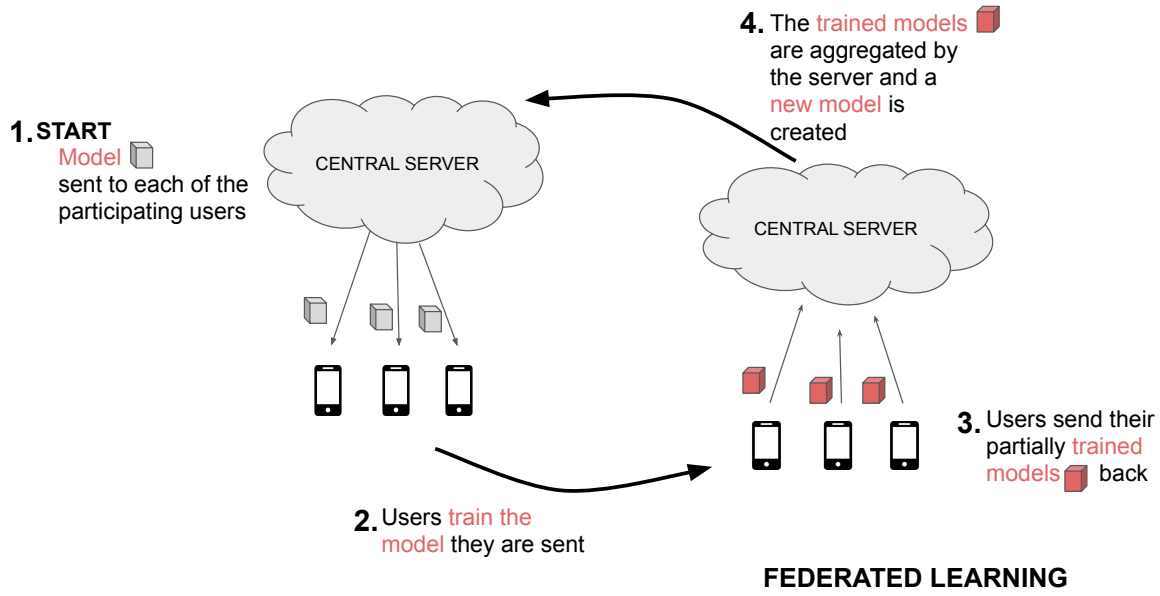


Figure 2.2: *Overview of Federated Learning*

federated learning libraries. The key difference between FedAvg and FedSGD is that FedAvg allows clients to train for multiple steps before sending the updated weights to the clients. FedAvg also initializes client weights to the same starting values when beginning training, which allows developers to naively average weights of models. McMahan et. al found that this approach worked well on the MNIST dataset [3].

FedProx. FedProx accounts for some of the assumptions made by FedAvg. It allows devices to submit partial work to the main model by introducing γ-inexact solutions and what the authors refer to as a *proximal term*. In short, the smaller the γ value, the more the device was able to train on the data and the more accurate the weights are considered. Meanwhile, the proximal term restricts updates to the main model based on how much the result obtained by a client differs from its original model. This also allows for partial work because along with the γ value, partial work never affects the main model as much as complete work does. The authors of FedProx consider FedAvg to be a special case of FedProx, which would mean that FedProx could be easily implemented in existing frameworks [1].

There are a variety of other algorithms which build on the work of the ones we have discussed, but this selection includes the most popular baseline algorithms implemented in federated learning frameworks [6].

2.1.2 Federated Learning Categorization

Federated learning algorithms are categorized by how data is grouped during the federated learning process. There are two broad categories: vertical federated learning, and horizontal federated learning. Each one approaches how data should be subdivided differently, and applies distinct additions in order to address the various considerations of federated learning in practical applications. In this project, we will focus on horizontal federated learning algorithms, but we describe both types here in order to emphasize what separates horizontal federated learning from vertical federated learning, and why we have chosen horizontal.

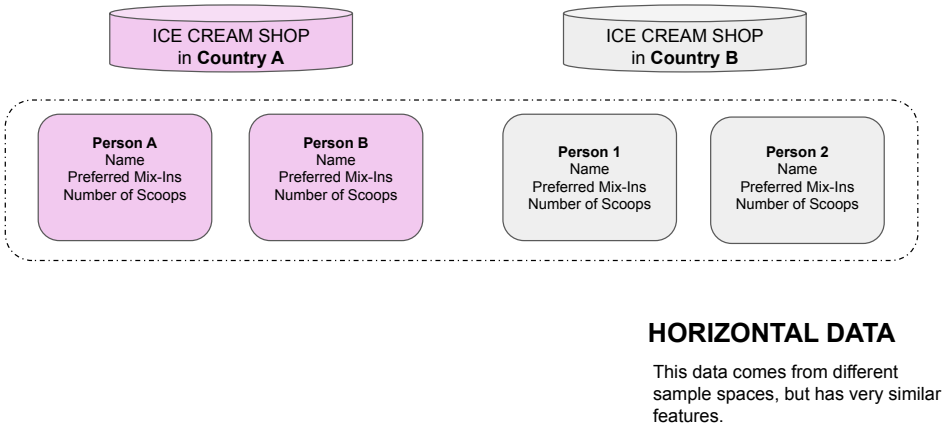


Figure 2.3: *Horizontal Federated Learning*

Horizontal Federated Learning Horizontal federated learning (shown in figure 2.3), also known as sample-based federated learning or homogenous federated learning, involves separating the data that has the same features but operates within a different sample space [8]. This means that data is separated by the types of characteristics the user has. This could involve separating out certain subgroups of device IDs in order to aggregate a subset of models [8]. This is analogous to training models on all data related to ice cream shops, regardless of where they are in the world. All the shops have approximately the same features, but their customer bases would be a very widely varied sample.

Vertical Federated Learning In vertical federated learning (shown in figure 2.4, also called feature-based federated learning or heterogenous federated learning, data shares the same

sample space, but different feature space [8]. In other words, data comes from similar users but has different characteristics. To continue our analogy, this would be like examining data from an ice cream shop and a build-your-own pizza restaurant in the same city. The two businesses may have similar enough goals that creating shared algorithms may be useful, and they share similar target groups (because they are in the same city), but the actual features of their businesses and transactions may be very different. Because they are two different businesses, their data should remain separated and distinct, and may even be managed in entirely different ways. In the real world, this approached has been utilized in banking [2]. Due to the unusual circumstances of two businesses sharing data to obtain a heterogenous perspective, this approach is less common in industry [2].

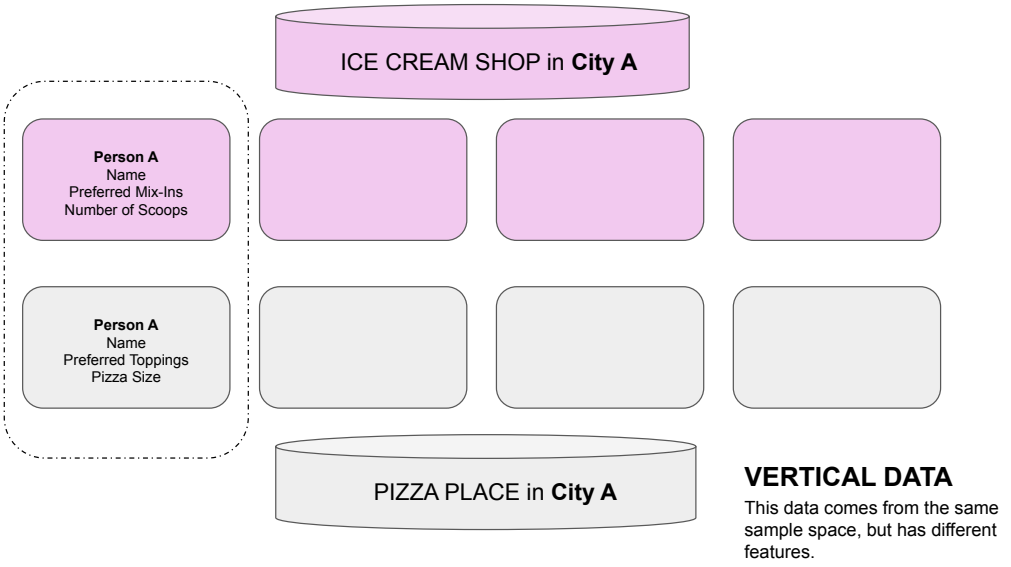


Figure 2.4: *Vertical Federated Learning*

In Section 2.1.1, we discussed a selection of early federated learning algorithms. FedProx, FedAvg, and FedSGD are all examples of horizontal federated learning. Horizontal federated learning is the most common form of federated learning, due to the fact that federated learning within industry contexts often relies on similar features. This also means that this form has a broader body of work dedicated to improving it through the key considerations discussed below [9]. We have decided to focus our scope on this type of federated learning.

2.1.3 Key Considerations of Federated Learning

According to Li et. al [1], the primary areas of focus in improving the field of federated learning are privacy, communication efficiency, statistical heterogeneity and system heterogeneity. These

areas affect the ability of the algorithm to perform at an optimal level, either in data transit or the complexity of computation. In this project, we will consider communication efficiency, statistical heterogeneity, and system heterogeneity. We will also discuss how privacy could be considered in the different algorithms we test.

Privacy Privacy is a prime concern of federated learning, and it is to some extent built into the field [3]. Although federated learning does definitionally offer some privacy, applications can leverage improvements in order to provide more. User information can be vulnerable on devices, and even the trained models can be vulnerable both in transit and on the server [3]. In order to add privacy, researchers have designed methods in order to further allow for user data and models trained on user data to remain private at all steps in the federated learning process. Differential privacy allows user's data to be further anonymized. Homomorphic encryption allows data to be used without being unencrypted. Secure multiparty encryption, or SMC, ensures each party knows nothing except the input and output directly relevant to them [10].

Communication Efficiency Communication efficiency is also central to federated learning applications, primarily because of cost [1]. As shown in 2.2, federated learning relies on being able to send and receive trained models between the central server and the users. Rounds of communication with the users are critical, but they should be minimized to the extent that they can without compromising the learning in order to minimize overall cost.

System Heterogeneity System heterogeneity is the concern that the computational power and communication abilities of mobile devices differ due to differences in connection strength, and hardware (for example, battery and computing level) [3]. As a result, federated algorithms must be able to tolerate device dropout, hardware discrepancies, and low device participation. The ability of a device to participate in training more than another leads to statistical heterogeneity.

Statistical Heterogeneity Statistical heterogeneity is the concern that data collected by users is "highly non-identically distributed" [1]. If the number of data points per device also varies greatly, this complicates the modeling of the problem and the finding of the solution. Li *et al.*mention that there are methods to combat this non-independent and identically distributed data such as training local models and metalearning, an approach which aims to learn about the learning process.

CHAPTER 3

RELATED WORK

3.1 Federated Learning Algorithm Development and Comparison

Federated learning algorithms are commonly evaluated in order to justify creation of a new federated learning algorithm. In this context, researchers begin by picking a common framework, and dataset in order to test. Next, they perform testing on an existing algorithm in order to display its shortcomings, and on their new algorithm in order to show how their new algorithm corrects them.

McMahan *et al.*demonstrated with their creation of FedSGD and FedAvg that federated learning is a practical method for deep learning. They describe their process moving from pure SGD across devices to FedSGD to FedAvg. Also, they found that by increasing computations per client and parallelism, the model converged faster and more accurately. In their conclusion, they hypothesized that with techniques such as differential privacy, more privacy could be guaranteed for clients [3].

Li *et al.*introduced FedProx in their paper, which they presented as a solution to the heterogeneity of federated data. They report that this is due to the proximal term and partial training allowance. Additionally, the authors establish guidelines for tweaking the hyperparameters of FedProx to maximize model performance; they also introduce adjusting hyperparameters during training based on the performance of the model. They also prove convergence for the algorithm under realistic settings [8].

Federated learning algorithms can be evaluated in order to determine which algorithm performs best according to different characteristics. Researchers Asad *et al.* [11] evaluate recent federated learning algorithms in terms of communication efficiency in order to highlight how to

improve communication efficiency for clients. Another research team, Wei *et al.* [12], designed a framework in order to evaluate federated learning algorithms in terms of privacy and protection from adversaries.

Our paper will evaluate federated learning algorithms using the consideration of statistical heterogeneity, and, similarly to researchers Asad and Wei, we will provide recommendations based on our findings in order to improve the algorithms we test.

3.2 Federated Learning Frameworks

Tensorflow Federated. Tensorflow Federated (TFF) contains two components to facilitate training and evaluation of federated learning models. The creators of TFF designed this framework with the intent to make experimenting with federated learning accessible even without deep knowledge of the underlying math and details. It was also designed to be extensible, so developers are constantly working on it and adding new features [13].

PySyft PySyft builds upon existing deep learning frameworks including PyTorch and TensorFlow[14]. Vitally, this is separate from Tensorflow Federated by cross-framework support. PySyft places an emphasis on secure and private deep learning using methods such as federated learning, differential privacy, and homomorphic encryption [2]. It is still under development and has an active Slack community.

PaddleFL PaddleFL is a federated learning framework based on PaddlePaddle, a deep learning framework produced by Baidu. The developers of PaddleFL divide the framework into two parts: compile time and runtime. Compile time focuses on selection of algorithms and generation of jobs, and runtime involves the actual training. They are also developing more vertical algorithms [15].

3.3 Datasets and Evaluating Federated Learning

LEAF LEAF is a benchmarking tool for federated learning. It contains datasets, statistics, and reference implementations for various federated learning algorithms including SGD, FedAvg, and Mocha as of Dec 2019. LEAF was created to allow federated learning researchers to learn about new solutions in their field. The creators continuously update LEAF with new algorithms, datasets, and metrics to help develop the field [16]. LEAF datasets include FEMNIST and CELEBA (discussed below). We focus primarily on the metric component of LEAF, as it is a way to compare different federated algorithms under varying conditions. For example, Caldas *et al.*describe the process of varying the sample number per user and the effects on the resulting accuracy. They also evaluate strain on the system by training on FEMNIST and logging the number of floating point operations per second (FLOPS) and data sent over the network during training.

FEMNIST FEMNIST (Federated Extended MNIST) is a set of datasets used for testing machine learning algorithms. It was created in 2017 by Cohen *et al.*as an alternative to the MNIST dataset, which was considered too easy for modern machine learning algorithms with published accuracies of 99.7%. While MNIST only contained the dataset for handwritten numerals 0-9, FEMNIST contains datasets for the written numerals as well as the letters of the alphabet. It also contains a dataset consisting solely of the MNIST dataset, so it could be used completely in place of the MNIST set [17]. We will use the FEMNIST superset of the MNIST dataset to evaluate the algorithms against statistical heterogeneity.

CELEBA CELEBA (large-scale celeb face attributes) is a second common dataset used for testing machine learning algorithms. It was created by Liu *et al.*, and contains 200k celebrity images. Its primary use is for binary classification testing whether the celebrity is smiling or not smiling. Although this is the primary use, this dataset also allows researchers to test other image classification problems; including but not limited to face characteristic recognition and face detection. The dataset is available exclusively for noncommercial research, and we utilize the CELEBA dataset in order to evaluate our algorithms.

CHAPTER 4

PROBLEM STATEMENT

4.1 Problem Statement

Our goal is to evaluate a subset of horizontal federated learning algorithms and assess their performance in relation to each other. We chose to focus our experiments on horizontal federated learning because this type of federated learning algorithm is more prevalent than its alternatives [9]. Our selection of algorithms is based on the critical baseline algorithms utilized in federated learning research. We will evaluate our chosen algorithms in terms of the three considerations: statistical heterogeneity, system heterogeneity, and communication efficiency. Our findings will also discuss how users can incorporate privacy through additions and how these additions may affect the other considerations, but this will not be the central focus of our exploration.

While research has already been conducted in evaluating certain aspects of federated algorithms [11], we hope to provide a baseline evaluation for critical federated algorithms and identify areas that the existing body of research has not yet explored. For example, Asad *et al.* [11] evaluated cutting edge federated learning algorithms in terms of communication efficiency. In our research, we evaluate the base federated learning algorithms on a broad spectrum. As a reminder, we evaluate not only communication efficiency, but also statistical heterogeneity and system heterogeneity. We take this stance so that we may provide recommendations for which algorithm to use for a certain consideration of federated learning. This also would allow beginners to begin their own research with a concrete understanding of where the strengths and weaknesses of each algorithms lie.

For our experiments, we leveraged TensorFlow Federated [13] (TFF) to provide a simulation of a federated learning environment. We make the assumption is that this simulation is an accurate representation of federated learning. We utilize TFF in our testing. Our goal is to provide the most broadly applicable recommendations possible for researchers and industry alike. TFF is widely

used in research, and widely referenced in federated learning work, making it a constructive framework in order to test our chosen algorithms. Although some more specialized frameworks exist, they are generally targeted at specific industries [9] or purposes[6], limiting their general applicability.

Our paper is successful if it provides well justified recommendations for federated learning algorithms for with respect to the conditions we have selected. Through our evaluations, we will demonstrate key benefits and deficiencies of the selected algorithms. We will provide well-rounded evaluation of why a user might select one algorithm over another, and how emphasis on statistical heterogeneity, system heterogeneity, and communication efficiency can make a difference in overall performance of federated learning.

CHAPTER 5

METHODOLOGY

5.1 Design

In this section we introduce our key design choices for our investigation, including implementations and testing details.

TensorFlow Federated We chose TensorFlow Federated because of its wide usage in the community. As discussed in our Related Works 3, TensorFlow Federated is a federated learning framework which allows researchers to experiment with federated learning without deep understanding of the underlying math [1]. TensorFlow Federated also benefited from widely available documentation. Throughout our preliminary search for what tools to utilize in this process, documentation was a key factor for selection because the team considered it critical to ensure the evaluation process was smooth. The TensorFlow Federated API and core allowed us to focus on critical differences between our chosen models, and implement them while using many of TensorFlow Federated's tools for federal learning.

Data We selected the EMNIST and CELEBA datasets as our baseline, and preprocess these datasets using the benchmark LEAF. As discussed in our Related Works, EMNIST is a widely used dataset for federated learning benchmarks. Notably, it is used by the creators of the models we are testing in order to prove the abilities of these federated learning algorithms. The specific dataset we have chosen to employ is FEMNIST. FEMNIST is an image classification dataset which is commonly used in federated learning research. The classification problem for this dataset is to discern the digits 0-9 and letters a-z. An example of the difficulty of this problem is shown in the figure 5.1: although this is a subset of the class 'q', it is difficult to distinguish the bottom row of 'q's from eights. As our second point of reference to compare the federated

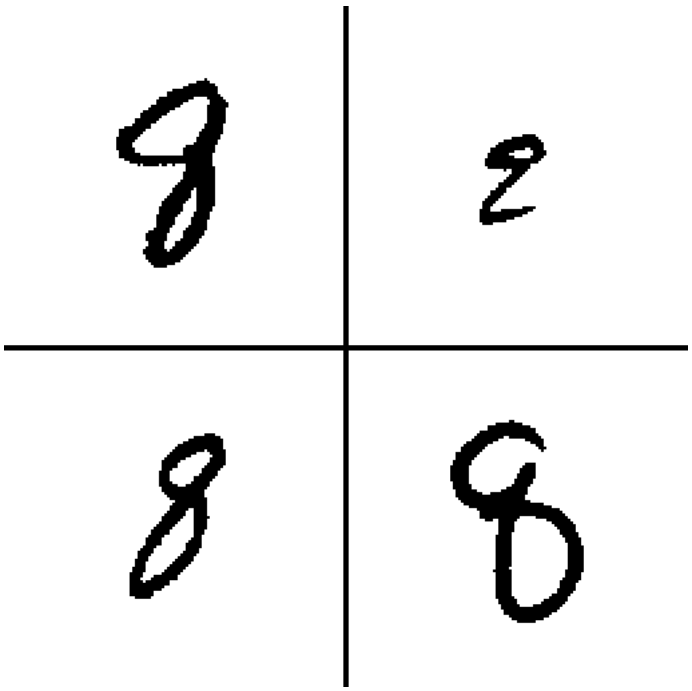


Figure 5.1: *Example of four samples from class 'q'*

learning algorithms, we selected the dataset CELEBA. CELEBA is an image classification dataset, modified by the federated learning benchmark LEAF (discussed in Chapter 3). The dataset consists of celebrity faces, and LEAF's preprocessing presents the binary classification problem of determining whether each face is smiling. CELEBA was a representative dataset for our purposes because image classification remains the central application of federated learning algorithms. In order to preprocess our data, we selected LEAF. As discussed in our related works, LEAF is a commonly accepted benchmark for federated learning and allows us to easily ensure our dataset has characteristics (for example, being independent and identically distributed) that are vital for our testing through their preprocessing flags.

Algorithms For Evaluation We approached our goal of evaluating a subset of horizontal federated learning algorithms by selecting two federated learning algorithms to perform our evaluation. The first algorithm we selected was FedAvg (discussed in Section 2.1), because, as previously discussed, FedAvg is one of the most popular and widely used federated learning algorithms. It is a common point of comparison for algorithms that have come after it [18]. We planned to utilize this algorithm as a point of comparison for our other selected algorithm. The second federated learning algorithm we selected for evaluation was FedProx (described in Section 2.1). We believe it is valuable to compare these two models because they are two of the most common federated learning approaches. FedAvg, as discussed in our background, is a special case of FedProx where all clients perform the same amount of work. FedProx is therefore

Figure 5.2: *Two examples from the CELEBA dataset with their labels*

a natural comparison with similar characteristics that would still provide notable differences for testing, evaluating, and comparison.

5.1.1 Our Pipeline

In order to execute our evaluation, we created a 6-step pipeline shown in Figure 5.3. The following steps are elaborated upon in the rest of this chapter. A step-by-step version of this pipeline can be found in Figure 7.2.

5.1.2 Docker

We have created a Dockerfile that downloads and installs all needed software to run our experiments. To run it properly, we suggest at least 8GB of RAM and 8GB of free disk space. Docker creates an isolated environment so that we do not have to worry about updates breaking any of the experiments. Also, by changing one line in the Dockerfile, we could upgrade or downgrade a library as needed.

5.1.3 Data and Preprocessing

LEAF We utilized LEAF in order to download and preprocess the FEMNIST and CELEBA datasets. LEAF downloaded and partitioned the data to create different distributions of the dataset, both independent and identically distributed and non-independent and identically distributed versions. This distribution creation was accomplished through LEAF's preprocessing flags and is described below. In addition, we utilized the preprocessing flag in order to sample different amounts of the dataset. We did this to better evaluate each algorithm in our project through testing with datasets representing different characteristics common to federated learning benchmarking.

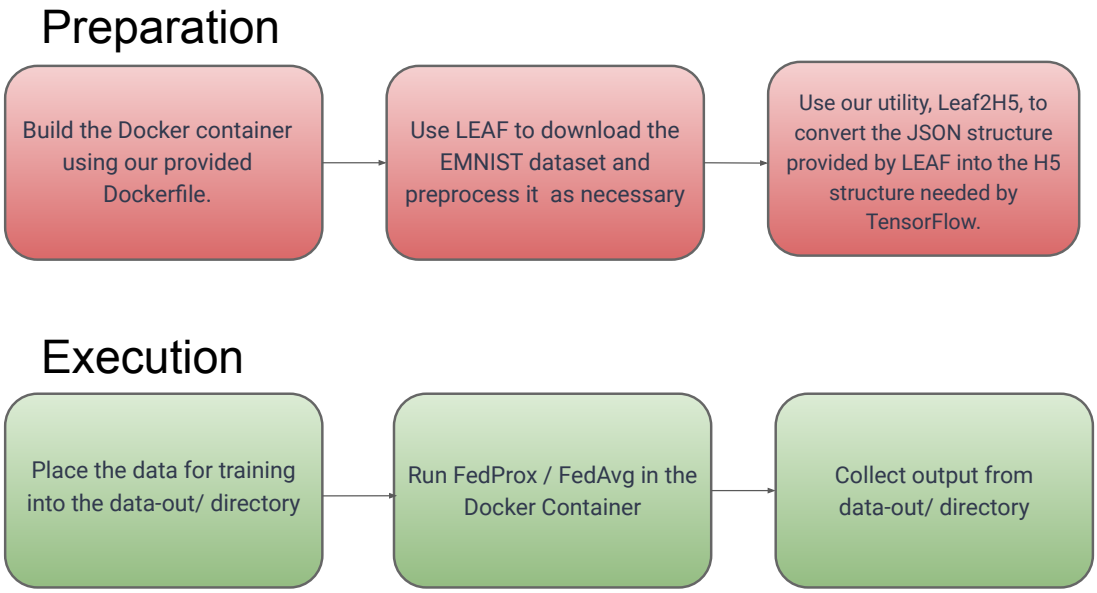


Figure 5.3: *Our Federated Learning Pipeline*

Generating our Dataset- FEMNIST We use LEAF to download FEMNIST, which initially contains raw image data. LEAF preprocesses these images into JSON files containing the data. Afterwards, we can further preprocess these JSON files to distribute them into train and test directories. We use the preprocess script provided by LEAF with the following command: `./preprocess.sh -s [n]iid -sf [0.1, 0.5, 0.75] -k 0 -t sample -smplseed 1549786595 -spltseed 1549786796`. More details on the meaning of this command appear in 7.2.

Generating our Dataset- CELEBA For CELEBA, we followed a similar process. Due to CELEBA being larger in size than FEMNIST, LEAF did not support the direct download of the dataset with their utilities. Therefore, we had to download the dataset ourselves and place the relevant files into LEAF's directories and finally run the preprocessing script. From there, we run the following command: `./preprocess.sh -s [n]iid -sf [0.1, 0.5, 0.75] -k 5 -t sample -smplseed 1549786595 -spltseed 1549786796` The command follows the same conventions as the one for FEMNIST above.

Conversion to HDF5 We created our own utility in order to convert each dataset to HDF5. This was necessary because the datasets output by LEAF were in .JSON format. However, TFF requires datasets in HDF5 format. Therefore, we took steps to create a simple utility in Python that parses the dataset from the JSON files and format it according to the example file that TFF uses in their sample code.

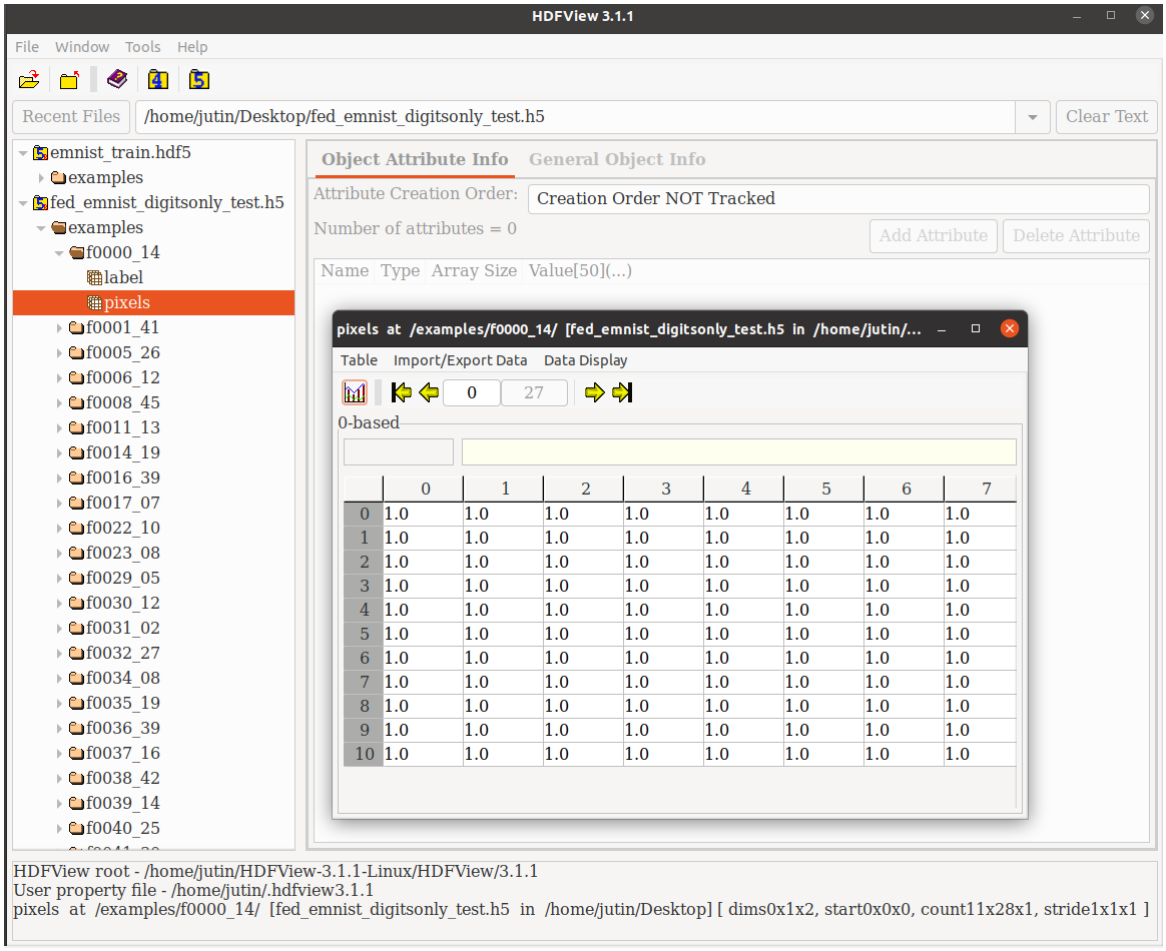


Figure 5.4: *HDFView In Action*

HDF5 and HDFView We examined the schema of existing dataset files from Google in order to determine how to format our own distribution of the datasets. We used HDFView to accomplish this. By using HDFView, we determined that the H5 schema is as follows:

root examples group
- user1 subgroup
  - pixel dataset
  - label dataset
- user2

How to Obtain Our Final Datasets In order to obtain the final EMNIST datasets we utilized for our training, we first generate the EMNIST dataset with LEAF, and distribute it either IID or NIID with LEAF's available preprocessing flags. We also would alter the fraction of data sampled to 0.1 (the default), 0.5, or 0.75. Then, we pipe the dataset through our utility, Leaf2H5, to obtain HDF5 files that are compatible with the Tensorflow training loop. Once we have the files output

by our utility, the first type dataset is ready to be used in training. In order to obtain the final CELEBA datasets we utilized for our training, we first generate the CELEBA dataset through the preprocessing scripts provided by LEAF, and distribute it either IID or NIID. We also would alter the fraction of the dataset to 0.1 (the default), 0.5, or 0.75. Then, we pipe the dataset through our utility, Celeb2H5, to obtain HDF5 files that are compatible with the Tensorflow training loop. Once we have the files output by our utility, the second type of dataset is ready to be used in training.

5.1.4 Implementation of FedAvg

The implementation of FedAvg we utilized is based on Tensorflow Federated's simple FedAvg implementation. Our updates allow the simulation to have a set percentage of stragglers in order to compare this implementation with FedProx. Tensorflow Federated's FedAvg includes SGD as the client and server side optimizer. The default learning rates are 0.1 and 1 respectively.

Algorithm 1 FedAvg (pseudocode adapted from Li. et. al [18])

for all Rounds **do**
  Server selects a subset of devices at random
  Server sends w to all chosen devices
  Each device updates wt using SGD to obtain updated weights
  Each device sends w_k^{t+1} back to the server
  Server aggregates the w's
end for

5.1.5 Implementation of FedProx

In order to implement FedProx, we made alterations to our existing implementation of FedAvg per the specifications of Li et al. This involved two critical changes: a new client side optimizer, and the introduction of a proximal term.

Client-Side Optimizer One of the key differences between FedAvg and FedProx is the client-side optimizer. Our changes to FedAvg's client-side stochastic gradient descent Keras optimizer involved altering 'resource_apply_sparse()', 'resource_apply_dense()', and _create_slots including the V* parameter in order to match the perturbed gradient descent optimizer described in Li et al. [18] Additionally, we had to override 'get_config' and '__init__() to create this Keras optimizer.

Proximal Term Scaling and (μ) Selection Within the client-side optimizer for FedProx, the gradient is updated using a function called the proximal term. This term, as defined by Li et. al, involves a normalization function scaled by a value called mu. This mu value may be altered to fit the model and dataset the optimizer is prepared for. Following the suggestions of Li et. al, we selected a μ for each dataset from a set list of [0.001, 0.01, 0.1, 0.5, 1]. To find the optimal μ for

the EMNIST dataset, we performed an exhaustive search on each of the provided values. This evaluation is described in 6. We compared the converging speed across all μ and found that 0.001 worked the best for the dataset on both IID and NIID data.

Algorithm 2 FedProx (pseudocode adapted from Li. et. al [18]

for all Rounds **do**
Server selects a subset of devices at random
Server sends w_t to all chosen devices
Each chosen device finds a w_k^{t+1} which is a γ_k^t -inexact minimizer of the loss function.
Each sampled device sends w_k^{t+1} back to the server
Server aggregates the w's for that round
end for

Chapter 6

Results

6.1 Investigation

Our preliminary investigation included tests for variable numbers of rounds while manipulating the μ value of the FedProx optimizer, whether the dataset was independent and identically distributed, the size of the dataset, and how our two models performed in comparison to each other. We then introduced stragglers and saw how the presence of them affected the convergence of each model.

6.1.1 Correctness

The first step in our project was to ensure our algorithms were implemented correctly and that our evaluation could consider their performance representative of the selected algorithms. We accomplished this by comparing the performance of our algorithms on EMNIST with the data provided in the FedProx and FedAvg papers. We found comparable results between our implementation and the results published, so we considered our implementations correct on the code level. For proofs of FedAvg and FedProx, see the related papers [1] and [3]. Our FedAvg and FedProx performance is shown in our quantitative studies.

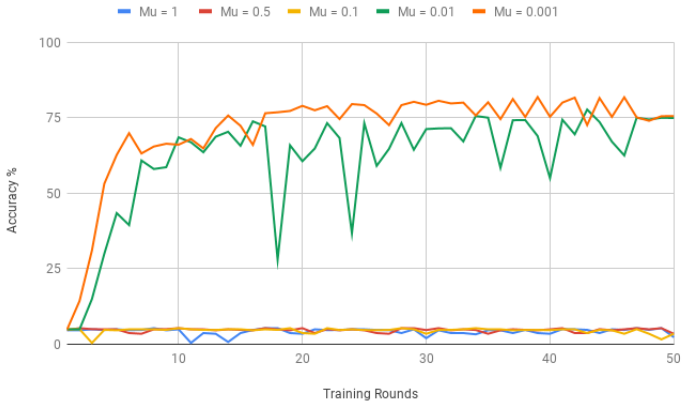
6.1.2 Overall Results

We show that under the circumstances we test, FedProx outperforms FedAvg in identically and independently distributed (IID) scenarios. In non-identically and independently distributed (NIID) scenarios, where statistical heterogeneity is increased, the relative performance of the algorithms is less predictable. For example, we found that FedAvg converges 6% faster than FedProx on CELEBA with no stragglers, while there was less than 1% difference between the

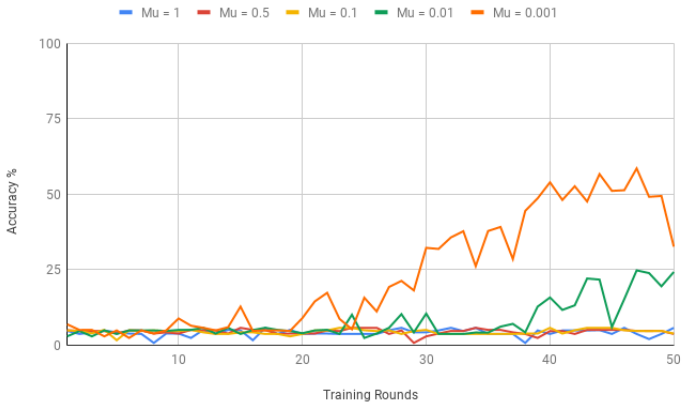
two on IID. In scenarios with stragglers (where system heterogeneity is introduced), FedAvg's was negatively impacted performance while leaving FedProx mostly unaffected in the MNIST dataset. When the same challenge is introduced with the CELEBA dataset, FedAvg was not able to perform at the same or better level than FedProx; FedProx converged in 50% of the rounds it took FedAvg to converge. However, NIID data again proved unpredictable with FedAvg performing 23% better than FedProx.

6.1.3 Quantitative Studies for EMNIST

The first step in our studies was to adjust the mu value for FedProx. As previously discussed, the mu value scales the proximal term (the client-side local solver). It must be adjusted according to the dataset, per the instructions of Li *et al.*[18]. In order to adjust the mu values, we performed tests of all potential mu values with both IID data, shown in Figure 6.1. We compared the converging speed across all μ and found that 0.001 worked the best for the dataset on both IID and NIID data.



(a) *FedProx Performance on FEMNIST IID Data, varying Mu*



(b) *FedProx Performance on FEMNIST NIID Data, varying Mu*

Figure 6.1: *FedAvg and FedProx Performance varying Mu on MNIST Data*

In Figure 6.2, we can see that when running on non-independent and identically distributed data (NIID) with stragglers enabled, FedProx converges more quickly and slightly more accurately then FedAvg. This is expected. FedAvg is unequipped to deal with the system heterogeneity that straggling devices introduces. Instead of incorporating partial training into the model, FedAvg simply drops under performing clients. FedAvg, on the other hand, is able to incorporate these straggling clients through its modified optimizer and training process. These clients allow FedProx's performance to advance more quickly, and ultimately be more accurate faster.

Our FedProx tests (shown in Figure 6.6 and Figure 6.5) show that FedProx is able to converge at approximately the same rate regardless of straggler presence. In both independent and identically distributed and non-independent and identically distributed scenarios, FedProx is well equipped to handle the system heterogeneity introduced by straggling clients.

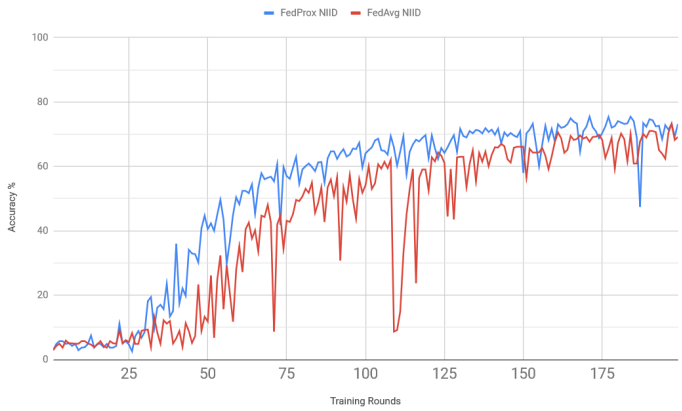


Figure 6.2: *FedAvg and FedProx Performance on NIID Data with Stragglers*

In Figure 6.3, we can see the result of running FedAvg and FedProx on non-independent and identically distributed data with no stragglers. In contrast to Figure 6.2, both algorithms perform approximately the same. This is expected because FedAvg is a special case of FedProx that is unequipped to handle stragglers, and so once stragglers are removed, these two algorithms treat their data in similar ways. The accuracy increases relatively slowly for both algorithms because NIID data introduces statistical heterogeneity, and this, as discussed in Section 2.1.3 is more difficult to train with.

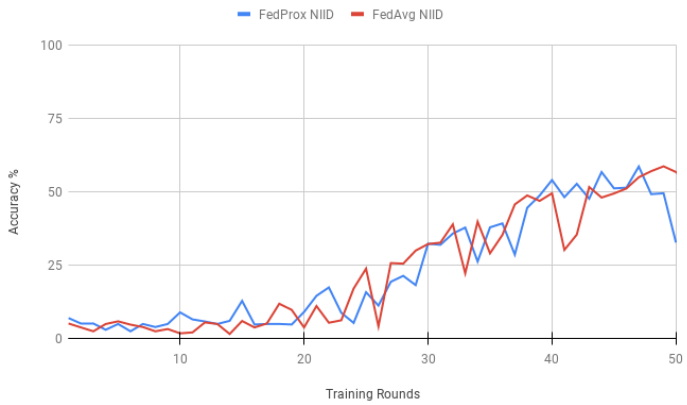


Figure 6.3: *FedAvg and FedProx Performance on NIID MNIST Data, no Stragglers*

In Figure 6.4, we can see the result of running FedAvg and FedProx on independent and identically distributed data. Both algorithms perform approximately the same, as expected. The accuracy increased for both of them much more quickly than with NIID data because IID data removes the statistical heterogeneity of NIID data.

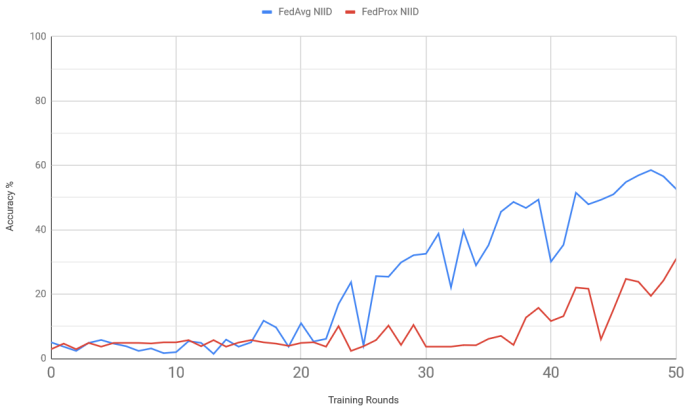
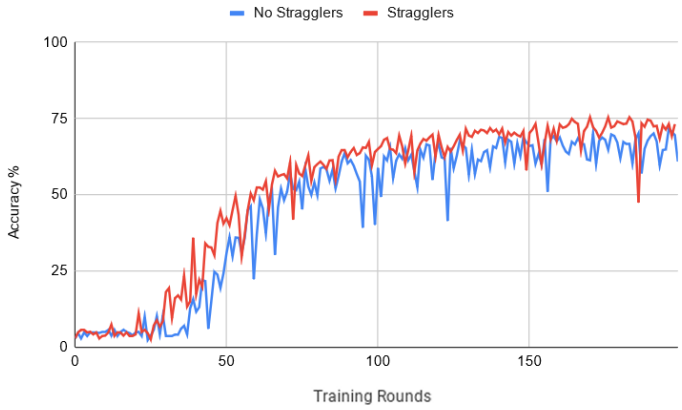
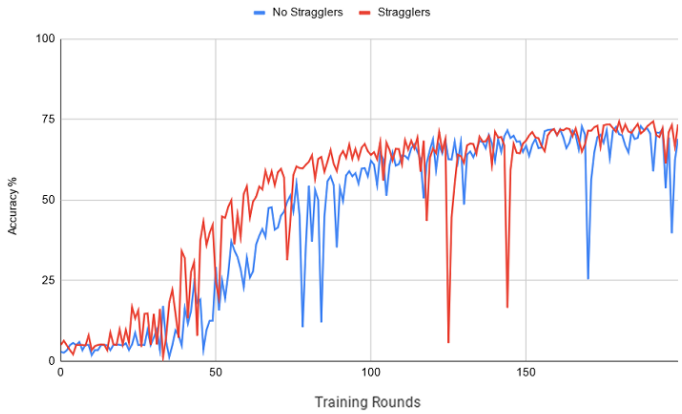


Figure 6.4: *FedAvg IID vs FedProx IID on MNIST Data, no Stragglers*



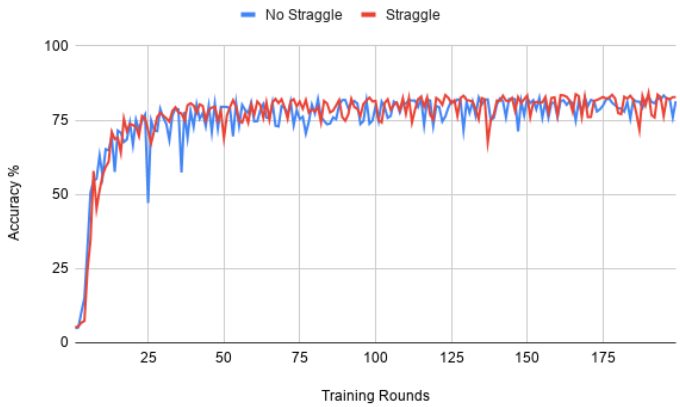
(a) *10 Percent of Dataset*

(b) *50 Percent of Dataset*

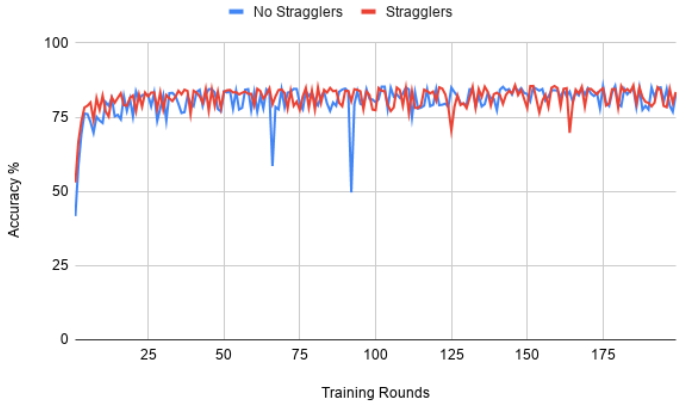


(c) *75 Percent of Dataset*

Figure 6.5: *Charts of FedProx Performance on NIID FEMNIST Data, with stragglers and varying data sizes. FedProx is able to handle the stragglers reliably.*



(a) *10 Percent of Dataset*



(b) *50 Percent of Dataset*

Figure 6.6: *Charts of FedProx Performance on IID FEMNIST Data, with stragglers and varying data sizes. FedProx is able to handle the stragglers reliably.*

6.1.4 Quantitative Studies for CELEBA

To begin working with this new dataset, the first step was again to select the appropriate mu value for FedProx. We tested five different mu values: 0.001, 0.01, 0.1, 0.5, and 1. As shown in Figure 6.7, the lowest mu value (0.001) converged the fastest and was therefore utilized as the mu value for CELEBA's indepenedent and identically distributed dataset for the remainder of testing. For CELEBA's non independent and identically distributed dataset, the mu value 0.001 appeared to converge more quickly initially, but was shown to have not yet converged. With 700 rounds of training, as shown in Figure 6.8, ultimately the mu value 0.001 converged most rapidly. We selected this value as our mu value for the non independent and identically distributed dataset.

Figure 6.7: *FedProx Performance IID CELEBA Data, varying Mu. Mu value 0.001 performs best.*

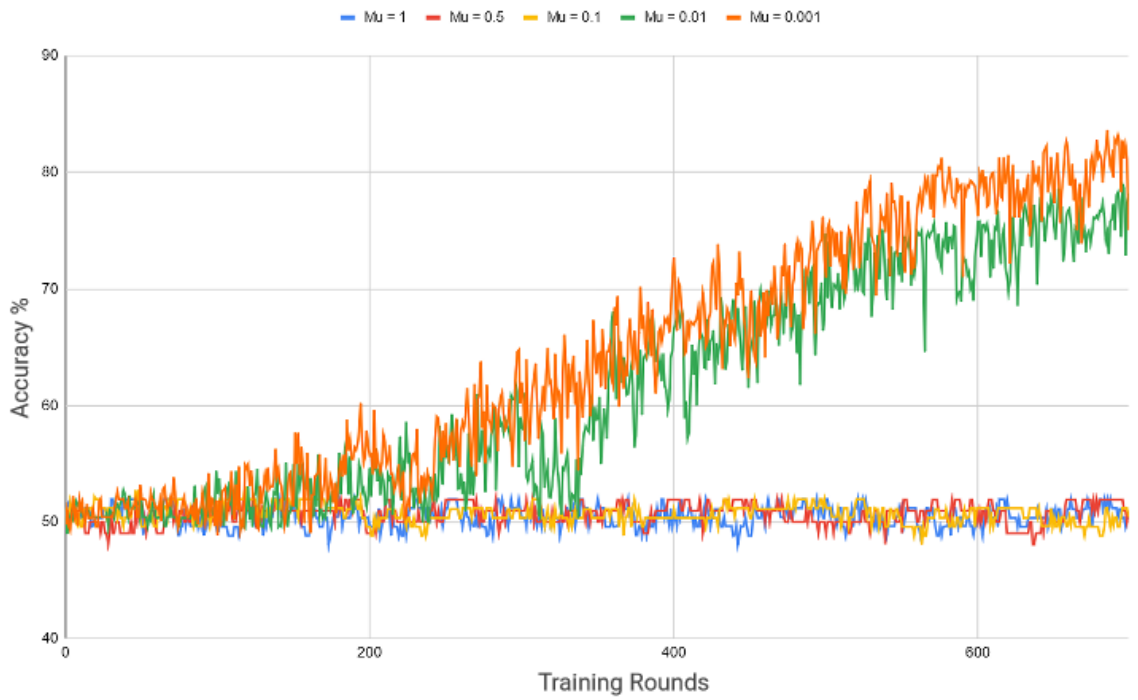
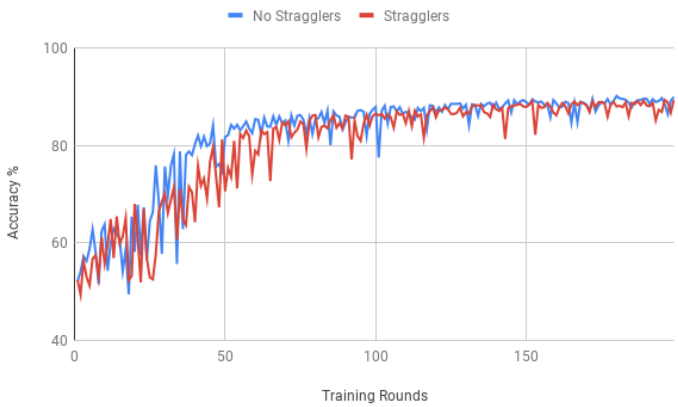


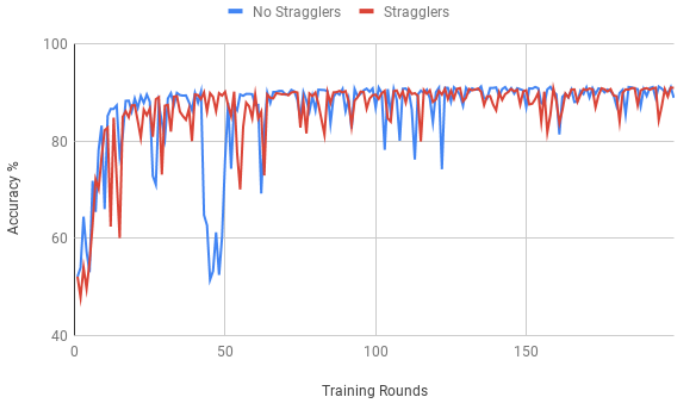
Figure 6.8: *FedProx Performance NIID CELEBA Data, varying Mu. This graph contains the averaged results over 5 rounds for each value. Mu value 0.001 performs best.*

When stragglers are introduced, FedProx performs better than FedAvg in all scenarios (as shown in Figures 6.9 and 6.10). As shown in these figures, we tested with our two distributions in three dataset sizes: 10 percent, 50 percent, and 75 percent. In our tests with the independent and identically distributed dataset, both FedAvg and FedProx with stragglers converged quickly and at nearly the same rate. However, FedProx converge slightly more rapidly. This is understandable because FedProx is intended to better accommodate stragglers than FedAvg.



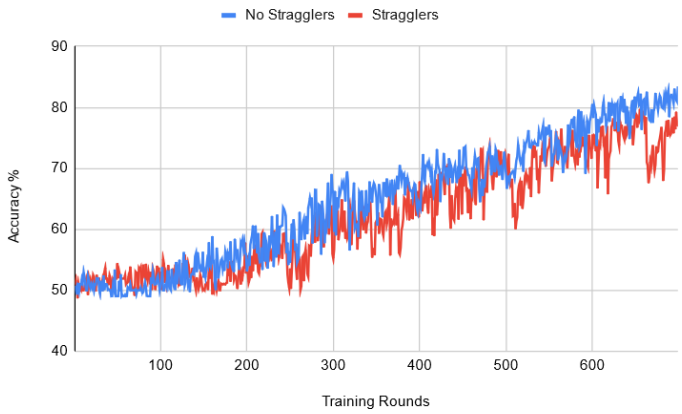
(a) *10 Percent of Dataset*

(b) *50 Percent of Dataset*

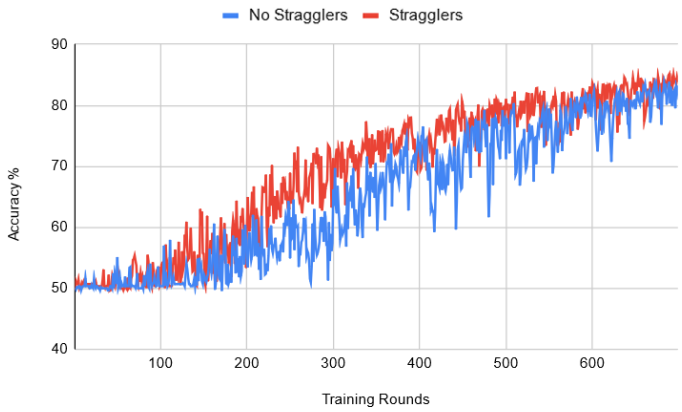


(c) *75 Percent of Dataset*

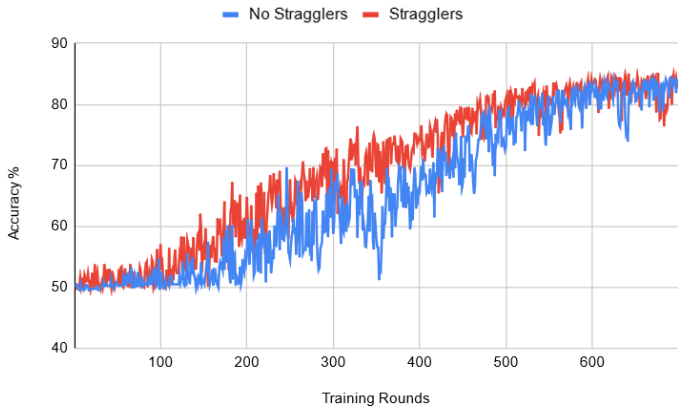
Figure 6.9: *Charts of FedProx Performance on IID CELEBA Data, with stragglers and varying data sizes*



(a) *10 Percent of Dataset*



(b) *50 Percent of Dataset*



(c) *75 Percent of Dataset*

Figure 6.10: *Charts of FedProx Performance on NIID CELEBA Data, with stragglers and varying data size*

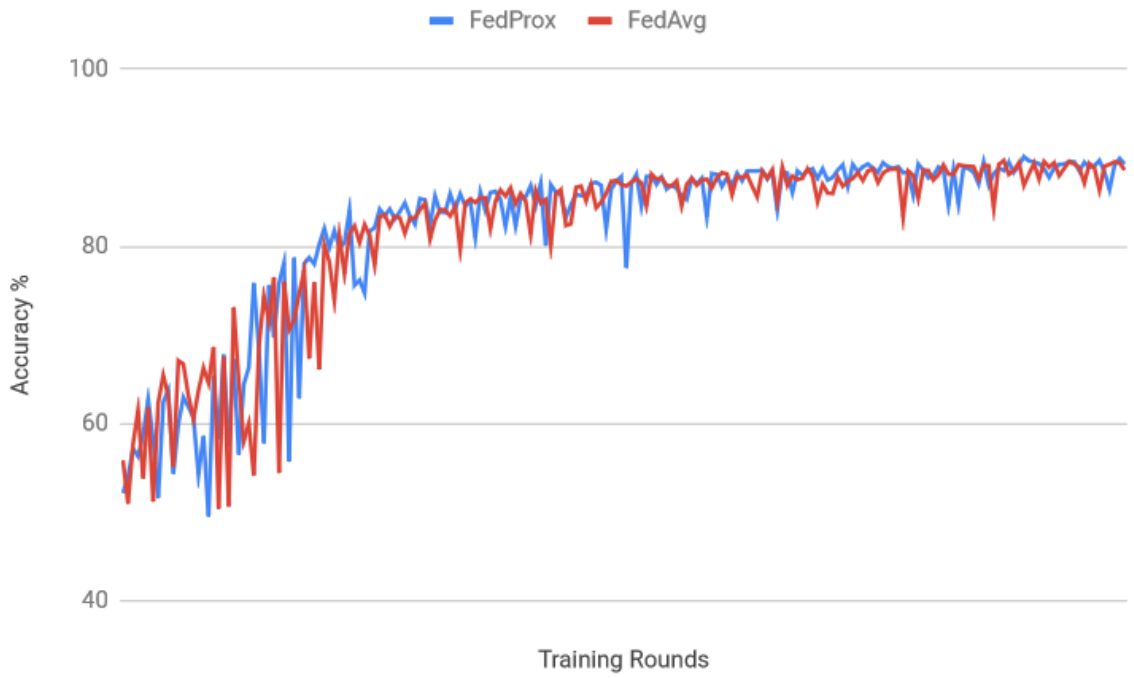


Figure 6.11: *FedProx Performance on IID CELEBA Data, Compared with FedAvg Performance*

FedProx performs approximately the same as FedAvg without stragglers and with an independent and identically distributed dataset, as shown in Figure 6.11. Although FedProx is better equipped for stragglers, both are able to rapidly converge given this distribution in the absence of stragglers.

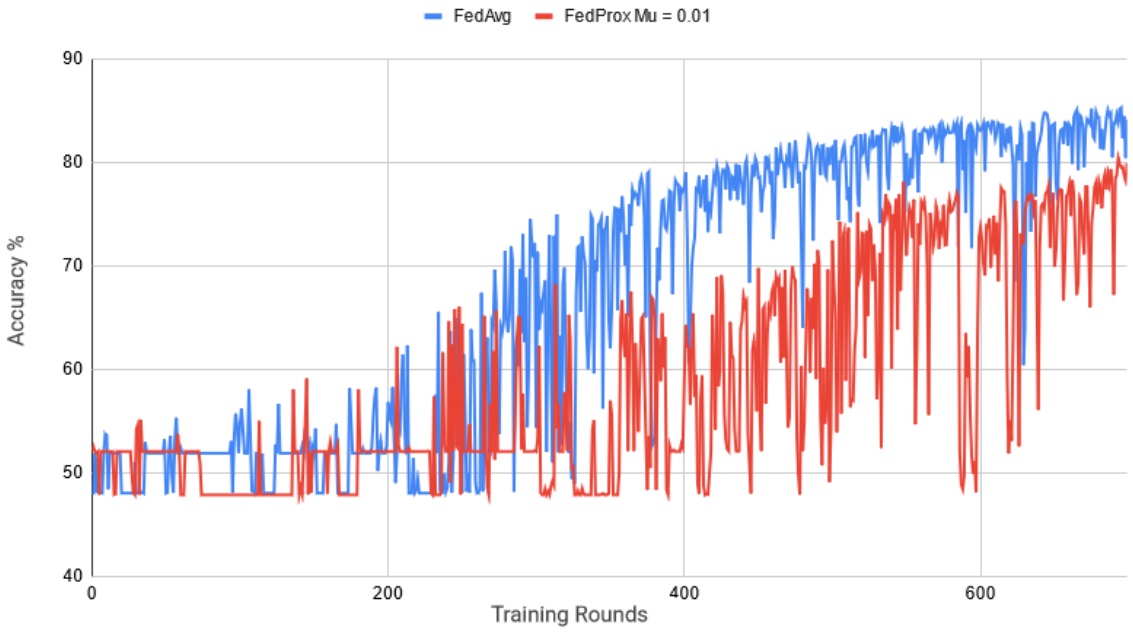


Figure 6.12: *Charts of FedProx Performance on NIID CELEBA Data, Compared with FedAvg Performance*

Without stragglers and training with a non independent and identically distributed dataset, FedAvg did perform better than FedProx in our experiment. The increase in accuracy on this graph is likely a product of the NIID distribution of the dataset. FedProx accounts for devices which may be under performing by scaling how their trained models are incorporated into the overall trained model, and this cautious approach to updating the model may be the cause of FedAvg converging faster.

Ultimately, both FedAvg and FedProx are appropriate choices for simple federated learning scenarios, depending on the situation. FedProx is better suited for scenarios where straggling devices are expected. FedAvg is better when no devices are straggling but the distribution of the data is such that the scaling employed by FedProx limits the training of the model. Either FedAvg or FedProx can be employed when there are no stragglers and the dataset is independent and identically distributed (though this is unlikely to happen in practice, with datasets not standardized to be federated learning benchmarks).

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

7.1 Conclusions

In this project, we have evaluated the performance of two Federated Learning algorithms, FedAvg and FedProx, on two datasets under a variety of conditions. FedProx mostly outperformed FedAvg by a non-negligible amount.

Comparative Performance In this paper we demonstrated that under a variety of circumstances, FedProx outperforms FedAvg in both IID and NIID scenarios. This is true with and without the addition of stragglers; a vital consideration when selecting a federated learning algorithm for a given purpose. We tested both algorithms on two datasets: FEMNIST and CELEBA. Our survey of these two federated algorithms demonstrates that FedProx handles data that is non identically and independently distributed much better than FedAvg can. For data that is not independent and identically distributed, the distinction is less important. We found that FedProx with straggling clients outperformed FedProx on non-independent and identically distributed data, and this is likely due to the inherent randomness of federated algorithm client selection. If a poorly performing client is sampled in a bad batch, then the performance of the model will suffer as a result. This explanation could be verified by future researchers who could test these algorithms using data they can manipulate the independent and identically distributed nature of in degrees.

Proximal Term Selection Throughout our experiments, we found that smaller mu values are better for the overall performance of FedProx. The creators of FedProx maintain that mu values can and should be adjusted based on the dataset from their chosen options of [0.1, 0.01, 0.001 and 1]. In testing all mu values in a variety of scenarios with our two datasets, we found that 0.001 is

consistently the mu value that leads to the highest overall validation accuracy more quickly. Our chosen datasets are common federated learning benchmarks, but present two starkly different types of image classification problems. Although other datasets even more removed from the two we tested with may perform differently, our findings display that the a lower mu value performs better.

Deliverable: Pipeline This paper also provides a concrete pipeline for testing federated learning algorithms. We provide a means of preprocessing the datasets we used into a form (HDF5 files) that allows them to be easily utilized in the environments we have prepared. Our docker-based local testing means that our findings are easily replicate, and our transition to Google Colab for CELEBA allowed us to verify that our pipeline is operable not only outside of Docker, but in a platform widely available even to those with minimal available RAM or other constraints which render a local testing environment difficult or impossible.

7.2 Recommendations and Suggestions for Future Researchers

Although our evaluation was a strong beginning to a deep evaluation of horizontal federated learning algorithms, there are several significant possible improvements. We were unable to perform these additions due to the time limitations of our project, but we still view these steps as valuable.

We believe that this evaluation would be more robust if it included more horizontal federated learning models, and therefore was able to present a broader view of the benefits and drawbacks of potential approaches. Other researchers have compiled extensive lists of federated learning algorithms, from [19] to [9]. Future researchers could choose a subset of popular horizontal federated learning algorithms to implement within TensorFlow Federated in order to provide direct and meaningful comparisons. We could also attempt to add additional models to our current pipeline in C term. This would likely also be valuable if the selected algorithms could be compared in the context of a specific industry or problem.

A second potential improvement is the introduction of further privacy measures. Although federated learning does hold an inherent level of privacy due to its ability to keep client data with the clients, researchers have found avenues to build on that level of privacy[9]. These privacy measures discussed in our background 2.1.3 are a part of the key considerations when choosing certain federated learning algorithms over others. Our team chose to focus on horizontal federated learning because of its level of popularity, and this popularity has also meant that it is the type of federated learning algorithm most commonly given these additions. The evaluation could be strengthened by adding an additional privacy measure (for example, homomorphic encryption or differential privacy) to one of the algorithms we did test. Privacy measures could also be included in another federated learning algorithm added to our evaluation.

For the training of the models, a potential improvement for comparing algorithms is training them for much longer than 200 rounds in order to observe their convergence. instead of cutting off our training at 200 rounds when we first started our experiments, we could have improved the quality of our initial findings by training for more rounds. Because the sample TensorFlow Federated code does not implement a method of stopping training based on convergence, we believe that this would be a reasonable addition to the codebase to run more robust experiments.

Our evaluation could be furthered by including testing with a wider variety of datasets. We consistently used EMNIST and CELEBA in order to focus our evaluation on the considerations of statistical and system heterogeneity. Despite the value in this perspective, it is relatively narrow. We could have utilized other datasets, especially datasets already prepared for our pipeline (primarily other datasets from LEAF that can be easily preprocessed, or datasets which can be loaded into TensorFlow Federated directly). We also discuss other federated learning evaluation benchmarks in our Related Works section 3. Including more datasets not only could have presented a broader view, but could have also provided a perspective more tailored to specific applications. Evaluating our chosen algorithms with a wider array of datasets could have added more credibility and applicability to our evaluation.

There are undoubtedly more steps to further enrich our project, but we believe these are the most significant alterations that could be made in order to continue the work of our MQP.

BIBLIOGRAPHY

[1] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," aug 2019. [Online]. Available: http://arxiv.org/abs/1908.07873http://dx.doi.org/10.1109/MSP.2020.2975749

[2] OpenMined, "Openmined/pysyft," 2020. [Online]. Available: https://github.com/OpenMined/PySyft

[3] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2017. [Online]. Available: https://arxiv.org/pdf/1602.05629.pdf

[4] K. Hao, "How apple personalizes siri without hoovering up your data," Apr 2020. [Online]. Available: https://www.technologyreview.com/2019/12/11/131629/apple-ai-personalizes-siri-federated-learning/

[5] "Covid-19 open ai consortium," Aug 2020. [Online]. Available: https://owkin.com/covid-19-open-ai-consortium/

[6] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and Open Problems in Federated Learning," dec 2019. [Online]. Available: http://arxiv.org/abs/1912.04977

[7] F. D. McSherry, "Privacy integrated queries: An extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 19–30. [Online]. Available: https://doi.org/10.1145/1559845.1559850

[8] Y. Liu, X. Zhang, and L. Wang, "Asymmetrical Vertical Federated Learning," apr 2020. [Online]. Available: https://arxiv.org/abs/2004.07427

[9] Q. Li, Z. Wen, and B. He, "Federated learning systems: Vision, hype and reality for data privacy and protection," *CoRR*, vol. abs/1907.09693, 2019. [Online]. Available: http://arxiv.org/abs/1907.09693

[10] Qiyang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong, "Federated Machine Learning: Concept and Applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, 2019. [Online]. Available: https://arxiv.org/pdf/1902.04885.pdf

[11] M. Asad, A. Moustafa, T. Ito, and M. Aslam, "Evaluating the communication efficiency in federated learning algorithms," 2020.

[12] W. Wei, L. Liu, M. Loper, K.-H. Chow, M. E. Gursoy, S. Truex, and Y. Wu, "A framework for evaluating gradient leakage attacks in federated learning," 2020.

[13] "Tensorflow federated," 2020. [Online]. Available: https://www.tensorflow.org/federated

[14] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," nov 2018. [Online]. Available: http://arxiv.org/abs/1811.04017

[15] "Paddle developer documentation," 2020. [Online]. Available: https://paddle.readthedocs.io/en/7.x-1.x/

[16] S. Caldas, P. Wu, T. Li, J. Konecný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," *CoRR*, vol. abs/1812.01097, 2018. [Online]. Available: http://arxiv.org/abs/1812.01097

[17] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *CoRR*, vol. abs/1702.05373, 2017. [Online]. Available: http://arxiv.org/abs/1702.05373

[18] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," *CoRR*, vol. abs/1812.06127, 2018. [Online]. Available: http://arxiv.org/abs/1812.06127

[19] N. Rieke, J. Hancox, W. Li, F. Milletarì, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, S. Ourselin, M. Sheller, R. M. Summers, A. Trask, D. Xu, M. Baust, and M. J. Cardoso, "The future of digital health with federated learning," *npj Digital Medicine*, vol. 3, no. 1, p. 119, Sep 2020. [Online]. Available: https://doi.org/10.1038/s41746-020-00323-1

Appendix A: Step-By-Step Pipeline

1. Build the Docker container using our provided Dockerfile.
2. Use LEAF to download the EMNIST dataset.
3. Use our utility, Leaf2H5, to convert the JSON structure provided by leaf into the H5 structure needed by TensorFlow.
4. Place the HDF5 file in a mount directory where it can be seen by the TFF instance running in docker.
5. Run FedProx or FedAvg, pointing TFF to the dataset.
6. TFF outputs the data from the training into a CSV file.

Alternatively, the pipeline can be set up in a cloud computing service by running the above steps without the Docker container.

Appendix B: Pre-Processing Commands

```
preprocess.sh # Script provided by LEAF
-s [n]iid # Distribute dataset [non] independently and identically
--sf [0.1, 0.5, 0.75] # Sample 10 / 50 / 75% of dataset
-k 0  # Minimum of 0 samples per client
-t sample # Partition parts of user data into train-test split.
# The other option is user, which partitions entire users
#into either train or test.
--smplseed
--spltseed # Seeds for sampling and split of dataset.
# Set these to our values to get exactly our dataset.
```