Imperial College London

Imperial College London

Department of Electrical and Electronic Engineering

M.Eng. Electronic and Information Engineering

Final Year Project - Interim Report

Hierarchical Federated Learning with Client Mobility

Author:
Alessandro Serena
alessandro.serena16@imperial.ac.uk
CID: 01188591

Supervisor:
Dr Deniz Gündüz
Imperial College London
EEE Department
Information Processing and Communications Lab

Date: January 27, 2020

Contents

1 Project Specification

Many modern applications rely on models generated using Machine Learning (§2.1) techniques on large-scale datasets [5]. However, in cases when data is generated by edge devices (e.g. mobile phones, IoT smart sensors, wearable devices, etc.), it is often not possible or inefficient to harvest this information and send it to a central server to perform a centralized fitting of the model due to bandwidth constraints and privacy concerns.
In Federated Learning [13] (§2.2) the model is updated in a distributed fashion. Edge devices update the global model on their local datasets, and then send the updates to a central server that aggregates them generating a new global model.

Considering that large-scale wireless implementations of this process can involve millions of devices, it is trivial to see that having only one aggregation server constitutes a huge bottleneck due to wireless communication latency and processing time [1].
To solve this issue, Hierarchical Federated Learning (HFL) [1, 12] has been proposed (§2.3). In this setting, an intermediate layer of small cell base stations (SBSs) is introduced. The clients communicate with one of the local base stations, which then periodically send their local models to the macro base station (MBS) to generate a global model.

A common shortfall of [1] and [12] is the fact that both papers present HFL assuming that the clients are fixed and always communicate with the same SBS, and that the clients are equally distributed in the clusters. These assumptions clearly do not hold in reality, where FL is meant to be performed by mobile edge devices such as smartphones or wearables.
The aim of this project is hence to investigate how wireless HFL systems perform when clients are allowed to move and to communicate with different SBSs, and how the existing HFL algorithms can be modified to take advantage of client mobility.

Understanding the behaviour of a HFL system when clients are allowed to move is not only important from a communications or distributed systems point of view (e.g. how devices can connect to a new SBS when they change geographical zone, how SBSs should handle the connection/disconnection of a user, etc.), but it is also crucial with regards to the learning task. In facts, as presented in [1], SBSs collaborate together with the MBS in the inter-cluster model averaging because otherwise they would not have any knowledge learnt from datasets of clients that do not belong to their own pool of devices.
Intuitively, if clients are allowed to move and hence to carry their local datasets to different SBSs, the advantage of performing inter-cluster model averaging decreases. If all devices were moving and ended up in all the clusters, every SBS would have received updates based on all the data held by the clients and therefore would end up with the same model parameters without having to share its model with the MBS.
One intuition is that it might be possible to reduce the frequency of the inter-cluster model averaging (thus saving bandwidth) as devices move more on average. This will be tested during the experimental phase (§4.2).

The experimental data for the evaluation of the algorithms will be obtained through simulating an HFL system in which clients can move in between each round to adjacent cells (and hence communicate with a different SBS) depending on some probability distribution specific for each client. The main idea is to have different classes of edge devices with different mobility factors, mimicking real de-

3%	10%	3%
10%	48%	10%
3%	10%	3%

Table 1: Table of probabilities of a device to move to an adjacent cluster or to remain in the same.

vices. For example:

- High mobility devices$\longrightarrow$ smartphones, wearables, self-driving cars.
- Low mobility devices $\longrightarrow$ IoT devices that can be moved, laptops, tablets.
- Fixed devices $\longrightarrow$ fixed IoT sensors, security cameras.

From a mathematical perspective this translates to having a 2-dimensional distribution in a 3x3 matrix of the probability of the point to remain in the current cluster (the central element of the matrix) or move to an adjacent cluster (the external elements of the matrix). See Table 1 for an example.

The aim of the simulator will be to evaluate the algorithms on two benchmark tasks:

- object recognition (MNIST, CIFAR-10), as in [1];
- next-word prediction with a LSTM, as in [13].

Both IID and non-IID [6] data distributions among clients will be tested. *Large mini-batch SGD*(§2.1.A), *local SGD* or *post-local SGD*(§2.1.B) will be used by the clients to perform their local updates.

The comparison of the algorithm behaviour will be based on two metrics:

- number of **communication rounds** needed to converge;
- **latency** analysis of upload and download total times given a wireless communication channel.

2 Background

2.1 Machine Learning

The objective of Machine Learning , in the setting of a *supervised learning* task, is to find the parameters of a model that satisfies the an finite-sum objective function of the form

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{1}$$

in which $f_i(w) = l(x_i, y_i; w)$ represents the error of the prediction on input data (x_i, y_i) of the model with parameters w [3].

The model parameters (also called *weights*) are learnt from a training set by performing iteratively a two-step process called *gradient descent*.
In the first step, *forward propagation*, each training sample is input into the model and the output is computed by using the current model weights. The model output is then compared with the ground truth label of the training datapoint. The distance between these two values represents the loss (i.e. error) of the current set of weights[1].
The second step, *backpropagation*, consists in calculating $\nabla l(x, y, w_t)$, the gradient of the loss on the datapoint x with ground truth label y with respect to the current set of parameters w_t, and applying the following update to the weights

$$w_{t+1} = w_t - \eta_t \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \nabla l(x, y, w_t) \tag{2}$$

where $\mathcal{D}$ is the training set.
Since our aim is to reduce the loss, the formula above shows that, at each backpropagation step, the current weights are modified in the opposite direction of the average gradient information and the amount of variation is regulated by the *learning rate* η.[2]

Minibatch SGD. One of the most common variations of gradient descent optimization used in machine learning is *minibatch stochastic gradient descent* (minibatch SGD, also known simply as SGD). In minibatch SGD, the gradient descent step described in equation (2) is computed on a small batch of datapoints randomly selected from the training set (rather than on the entire dataset). To reuse datapoints in multiple minibatches, minibatch SGD goes over the entire training set more than once; each complete pass over the training set is referred as one *epoch*.
This method reduces significantly the computational cost of a backpropagation step because less derivatives have to be computed, and improves generalization accuracy.
It is necessary to note that the performance of this optimization technique is controlled by the newly introduced hyperparameters *minibatch size* and *number of epochs*.

2.1.A Large Minibatch SGD

Researchers in [4] have studied how to increase efficiency of minibatch SGD for distributed synchronized learning applications. When the learning task is split among multiple learners, using small batch sizes increases the number of batches per epoch and therefore the number of times that the learners have to communicate with each other. The results in [4] prove that it is possible to increase the batch size up to 8192 and that this

[1] There are various ways of computing the loss, depending on the task that we want the model to perform. Generally, *euclidean distance (L2-norm)* or *mean squared error (MSE)* are used to express the loss in regression problems, *categorical cross-entropy* for classification.

[2] In the expression provided η has a subscript t because the learning rate can change over iterations due to other hyperparameters such as *learning rate decay*, or techniques like *learning rate warmup* (§2.1.A).

reduces drastically the training time required. In their work, they present some hyperparameter tuning techniques that are necessary to achieve said results:

- **Learning rate scaling:** when the minibatch size is multiplied by k, multiply the learning rate by k.
- **Learning rate warmup:** following the expression in equation (2) we can compare the SGD weight vector update step for a small batch size (n) and a larger one (kn). In the first case, after k iterations the resulting weights are given by

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_|} \nabla l(x, w_{t+j}) \tag{3}$$

  While for a single SGD step with batch size kn we have

$$w_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_|} \nabla l(x, w_t) \tag{4}$$

  By setting the learning rate $\hat{\eta} = k\eta$ (following the scaling technique outlined in the point above) we see that the expressions (3) and (4) yield similar parameter vectors, and therefore a similar learning curve, only if we could assume that $\nabla l(x, w_{t+j}) \approx \nabla l(x, w_t)$. This assumption generally holds in most cases but when the batch size is scaled to very large values and at the beginning of training when the network is changing rapidly.
  To solve the latter problem, it is suggested to use a warmup strategy to adapt the learning rate in the first epochs of training. *Constant warmup*, used in [5] can be performed by setting a low constant learning rate for the first epochs[3] and then setting it to $k\eta$. This method, however, yields a similar training curve only for low ranges of batch size scaling factors k.
  For large k, it is advised to use a *gradual warmup*, where the learning rate is iteratively incremented from a low value to the desired $\hat{\eta} = k\eta$ during the initial epochs.
  After warmup the learning process goes back to the original learning rate schedule.

Other parameters that have influence on distributed SGD are *batch normalization* [8, 7], *weight decay* [14], *momentum correction* [11], *gradient aggregation* (find more about this in §2.2 and §2.3), and *data shuffling* (i.e. how a dataset is split and assigned to the distributed learners).

2.1.B Local SGD

Since using large minibatches in SGD can cause convergence issues and generalization problems, researchers in [10] have demonstrated that participants in a distributed learning system can achieve better accuracy and reduce communication rounds by performing locally several parameter updates with small batch sizes sequentially, before sharing their update.

They also discuss two variations of local SGD:

- **Post-Local SGD:** since in the initial phase of training weights change rapidly, gradient updates need to be communicated as soon as they are computed to achieve better convergence. In post-local SGD, the local SGD algorithm is only started after a few rounds of small minibatch SGD. This lets us

[3] In [4] the warmup phase encompasses the first 5 epochs of training.

take advantage of warmup strategies, as those described in §2.1.A.

- **Hierarchical Local SGD:** in a hierarchically organized distributed learning system, local SGD is applied to different layers of the architecture using different numbers of minibatch SGD iterations.

2.2 Federated Learning

Modern applications make use of models trained on large-scale amount of data. However, when this data is generated by edge devices (e.g. text that is typed on a smartphone keyboard, heartbeat information recorded by a wearable device), it is often impossible or inefficient to send it to a central server to be used in the model fitting, due to privacy concerns and bandwidth constraints.
For this reason, Google researchers have developed *Federated Learning* (FL) [13], an **algorithm to perform privacy-preserving distributed machine learning on the edge**.
Federated optimization (i.e. optimization in federated learning) presents several key properties that make it differ from distributed optimization:

- The training data is generated by a particular user acting on a certain device and hence the data of each device would be **non-IID**.
- Some users may use thier device more with respect to others, making the data generated **unbalanced**.
- Since FL is thought to be used on all kinds of edge devices, it is expected that the number of devices taking part in a FL system to be much larger than the number of data samples per user. The system can be said to be **massively distributed**.
- Edge devices have **limited communication**, as they can go offline and they use crowded or expensive connections.

In FL the learning task is handled by a loose federation of clients, coordinated by a central base station. The algorithm that controls FL is called *Federated Averaging* (FedAvg), and is described below and defined in algorithm 1.
The initial parameter vector is generated by the server. Then the algorithm proceeds in rounds all composed of the same steps:

1. A subset of the clients is chosen at random[4] to participate in the current learning round.
2. The latest parameter vector is distributed by the server to the selected clients.
3. Each client updates the model parameters by performing minibatch SGD on their local dataset.
4. The new weights are sent by each client to the server.
5. The server aggregates all the received updates by taking a weighted average

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{5}$$

Since the training data is never shared by the clients, it is intuitive to understand how this process preserves privacy.

[4] A more efficient way to select clients is presented in [15] and discussed in §2.2.A.

Algorithm 1: `FederatedAveraging` [13]
The K clients are indexed by k; each client uses minibatch size B, number of epochs E, and learning rate η.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

In the following subsections, we will describe some enhancements that have been made to the Federated Learning algorithm.

2.2.A Client Selection

In [13] the clients are selected randomly to participate in each federated averaging step. This causes the time necessary for each round to be strongly dependent to the random selection. To increase the efficiency of this process, researchers in [15] have added a `Client Selection` step at the beginning of each round before the server sends out the global model.

1. A fraction of clients is selected to participate in the `Client Selection` step.

2. Clients reply to the server with information about the time needed for the update and for the upload of the model.

3. The server chooses the clients to keep by constraining the maximum length of a round to a fixed value, while trying to maximize the number of clients chosen.

After the `Client Selection` step, the FL algorithm continues as described in §2.2.

2.2.B Secure Aggregation

Secure aggregation [2] is a cryptographic technique that allows the clients to submit their updates encrypted to an *aggregation server*, which is able to aggregate them without decrypting them. The combined update is then sent to the central server by the aggregation server.
This allows for increased security because theoretically it is possible to derive the training data that generated an update if the update vector can be seen unencrypted.

2.2.C Data Selection

[18] presents a technique for clients to perform data selection to decide which data to use when updating the model weights. This technique can positively influence the convergence rate, but mostly in cases when we deal with real-world data. Since in this project we are using benchmark datasets, the utility of implementing client-level data selection will be evaluated during the development of the simulator.

2.2.D SCAFFOLD

In [9] researchers present `SCAFFOLD`, a correction mechanism for weights updates designed for Federated Learning. This procedure attenuates the problem of local (client-level) updates drifting apart due to non-IID and imbalanced distribution of data across clients.

2.3 Hierarchical Federated Learning

Federated learning was envisioned to be used on large scale networks of devices. When these clients are in the range of millions, having a cloud-based system with only one aggregation server to which all clients send their updates represents a performance bottleneck. In *Hierarchical Federated Learning* [1, 12], we introduce an intermediate layer of small cell base stations (SBSs). Clients are grouped geographically around the closest SBS, to which they send their updates[5]. Periodically SBSs communicate their current model to a macro cell base station (MBS), which averages them and redistributes a new global model.

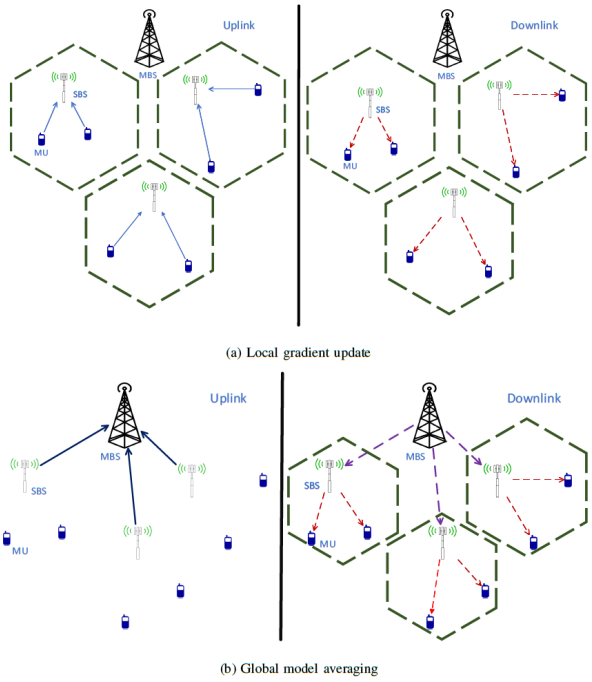

Figure 1: Hierarchical Federated Learning [1]

This process, together with gradient sparsification [17] reduces considerably the communication latency since it allows for carrier reuse in different clusters, without sacrificing model accuracy.

The algorithm described in [1] is presented below.

Algorithm 2: `Hierarchical Federated Averaging`

Initialize $\mathbf{W}$ and $\tilde{\mathbf{W}}$
for $n = 1, 2, \dots N$ **do**
  Initialize $\mathbf{W}_n$ and $\tilde{\mathbf{W}}_n$
for $k = 1, 2, \dots K$ **do**
  Initialize $\mathbf{w}_k$
for $t = 1, \dots T-1$ **do**
  Computation and Uplink:
  for $k = 1, 2, \dots K$ **do**
    Randomly select a mini-batch $\mathcal{I}_k \subseteq \mathcal{D}_k$
    Calculate $\mathbf{g}_{k,t} = \frac{1}{|\mathcal{I}^k|} \sum_{i \in \mathcal{I}^k} \nabla f_i(\mathbf{w}_t)$
    $\mathbf{u}_{k,t} = \sigma \mathbf{u}_{k,t-1} + \mathbf{g}_{k,t}$
    $\mathbf{v}_{k,t} = \mathbf{v}_{k,t-1} + \mathbf{u}_{k,t}$
    $g_{th} \leftarrow \phi^{ul}_{MU}$ of $|\mathbf{v}_{k,t}|$
    $mask \leftarrow |\mathbf{v}_{k,t}| \geq g_{th}$
    $\hat{\mathbf{g}}_{k,t} = \mathbf{v}_{k,t} \odot mask$
    $\mathbf{u}_{k,t} = \mathbf{u}_{k,t} \odot \neg mask$
    $\mathbf{v}_{k,t} = \mathbf{v}_{k,t} \odot \neg mask$
    send $\hat{\mathbf{g}}_{k,t}$ to associated SBS
  Model Average:
  for $n = 1, 2, \dots N$ **do**
    Update $\mathbf{W}_n(t+1) = \tilde{\mathbf{W}}_n(t) - \eta \hat{\mathbf{g}}_n + \beta_s \epsilon_n(t)$
  if t is divisible by H **then**
    for $n = 1, 2, \dots N$ **do**
      $\Delta_{\mathbf{W}_n}(t+1) = \mathbf{W}_n(t+1) - \tilde{\mathbf{W}}(h)$
      send $\Omega(\Delta_{\mathbf{W}_n}(t+1), \phi^{ul}_{SBS})$ to MBS
      $\mathbf{e}_n(t+1) =$
      $\Delta_{\mathbf{W}_n}(t+1) - \Omega(\Delta_{\mathbf{W}_n}(t+1), \phi^{ul}_{SBS})$
    $\Delta_{\mathbf{W}} = \sum_n \Omega(\Delta_{\mathbf{W}_n}(t+1), \phi^{ul}_{SBS}) + \beta_m e$
    MBS transmit $\Omega(\Delta_{\mathbf{w}}, \phi^{dl}_{MBS})$ to all SBSs
    $\mathbf{e} = \Delta_{\mathbf{W}} - \Omega(\Delta_{\mathbf{W}}, \phi^{dl}_{MBS})$
    $\tilde{\mathbf{W}}(h+1) = \tilde{\mathbf{W}}(h) + \Omega(\Delta_{\mathbf{W}}, \phi^{dl}_{MBS})$
    for $n = 1, 2, \dots N$ **do**
      $\mathbf{W}_n(t+1) =$
      $\tilde{\mathbf{W}}(h) + \Omega(\Delta_{\mathbf{W}}, \phi^{dl}_{MBS}) + \mathbf{e}_n(t+1)/N$
  for $n = 1, 2, \dots N$ **do**
    $\delta_{\mathbf{W}_n}(t+1) = \mathbf{W}_n(t+1) - \tilde{W}_n(t)$
    SBS_n sends $\Omega(\delta_{\mathbf{W}_n}(t+1), \phi^{dl}_{SBS})$ to MUs
    $\tilde{\mathbf{W}}_n(t+1) = \tilde{\mathbf{W}}_n(t) + \Omega(\delta_{\mathbf{W}_n}(t+1), \phi^{dl}_{SBS})$
    $\epsilon_n(t+1) = \delta_{\mathbf{W}_n}(t+1) - \Omega(\delta_{\mathbf{W}_n}(t+1), \phi^{dl}_{SBS})$
  Update:
  for $n = 1, 2, \dots N$ **do**
    for $k \in \mathcal{S}_n$ **do**
      $\mathbf{w}_k(t+1) = \tilde{\mathbf{W}}_n(t+1)$

[5] the learning process between clients within a cluster and their SBS works as described in §2.2

3 Implementation Plan

In the following subsections we will describe the structure and implementation plan of the simulator mentioned in §1.

3.1 System Structure

The main functionalities that are required from the simulator can be listed as follows:

1. **Configuration:** we should be able to set an initial state of the simulator and set rules for the simulation execution. See §3.1.A for more information about the configuration mechanism.

2. **Training:** the system should be able to train a model following the *Hierarchical Federated Averaging* algorithm (described in §2.3).

3. **Testing:** we should be able to test the accuracy of the fitted model with respect to a testing dataset, at all levels of the network (i.e. clients, SBSs, MBS).

4. **Logging:** the system should provide a log of its status at any time during the simulation execution. This is to be used for the analysis of the algorithms performance, described in §4.

5. **Move clients:** clients should be able to move across different clusters during a simulation.

From an Object-Oriented Programming (OOP) perspective the system can be implemented with the class model drawn in table 2. The `simulator` itself runs a list of `simulation`s. This design choice was made to allow for parallel execution of experiments. Since each of them can be run as a separate thread, we can speed up computation, also by using the multiple CPUs and GPUs of the IPC Lab servers.
Each `simulation` controls a `network` which is composed by a `MBS` (a `server`-type object) and a list of `cluster`s.
`cluster`s are made of one `SBS` (again, a `server` object) and a list of `client`s.

Some of the methods are common to most of the classes, as they implement the main functionalities described above.

The simulator will be coded in Python, in order to take advantage of the PyTorch[6] library. PyTorch is going to be used for the model training at client-level and for easily accessing and manipulating multiple open-source benchmark datasets (MNIST, CIFAR-10, WikiText-2, Penn Treebank, etc.).

3.1.A Configuration File

The configuration information will be contained in a `JSON` file. Each file would contain the following information:

- **Network structure:** number of clusters, number of clients per cluster, clients mobility rate (i.e. the mobility class type of the clients).

- **Model structure:** model to train (CNN, Resnet, LSTM), model weights initialization (random, from a file).

- **Data structure:** dataset to be used (MNIST, CIFAR-10, Penn Treebank, etc.), data distribution among clients (IID, non-IID).

- **Clients learning parameters:** algorithm (minibatch SGD, large minibatch SGD, local SGD, post-local SGD), other parameters (e.g. batch size, number of epochs, learning rate warmup, etc.).

[6] https://pytorch.org/

CLASS	ATTRIBUTES	METHODS
simulator	- simulations [List]	- start(simulation, config_file, log_file)
simulation	- config - network - log_file - round_number	- configure() - step() - log()
network	- MBS [server] - clusters [List]	- learn() - log() - move_clients()
server	-id - model	- log() - test(test_set)
cluster	- SBS [server] - clients [List]	- learn() - log()
client	- id - model - dataset	- learn() - log() -test(test_set)

Table 2: Class model outline.

- **Servers learning parameters:** inter-cluster (global) update frequency (MBS), intra-cluster update frequency (SBSs).
- **Simulation structure:** stop condition (after N rounds, after reaching x% test accuracy).
- **Logging directives:** logging frequency, logging verbosity, output file to write the log to.
- **Terminal output directives:** `STDOUT` output frequency, verbosity level.

When a parameter is not specified in the configuration file, the simulator will try to use a default value.

3.2 Milestones and Versions

The simulator will be built through the development of different, incremental, versions. The milestones that I will work towards are:

1. **MVP[7]:** At this stage the simulator should be in its most simple working state. It should be able to start a simulation as described in a configuration file, train with HFL a simple model and test the model accuracy on a separate test set. The system should be able to log the status of the system at any time during a simulation.
   Devices are not expected to move among clusters in this version.
2. **MOBILITY:** Include support for client mobility.
3. **LEARNING TASKS:** Add support for more complex learning tasks (next-word prediction), models (Resnet, LSTM) and data distribution across clients (IID, non-IID).
4. **LEARNING METHODS:** Add support for more complex client-level learning algorithms (large minibatch SGD, Local SGD, Post-local SGD).

[7]Minimum Viable Product

#	TITLE	CONFIGURATION	TRAINING	TESTING	LOGGING	MOVE CLIENTS
1	MVP	Config mechanism working. Basic JSON structure.	- Only simple CNN for MNIST w/ IID data among clients. - Client-level training done with minibatch SGD. - Hierarchical training structure implemented.	Testing infrastructure working for MNIST.	Top-down logging functionality implemented.	No mobility.
2	MOBILITY	Include mobility information in - configuration file - classes hierarchy	-	-	-	Add support for mobility functionality
3	LEARNING TASKS	Add support for selection of learning task of simulation - Model - Dataset - Data distribution across clients	Include new models	Add testing support for new datasets learning tasks	-	-
4	LEARNING METHODS	Add support for selection of client-level learning algorithm	- Large minibatch SGD - Local SGD - Post-local SGD	-	-	-
5	EXPERIMENTS	-	-	-	-	-

Figure 2: Outline of the project milestones.

5. **EXPERIMENTS:** At this stage the simulator development is complete.
   Experiments will be conducted as described in the evaluation plan (§4).
   Ideally, it would be useful to have a tool to automate the experiments execution and one to visualize and compare the results of different simulations.
   However, the simulator functionalities will be expanded or modified if new experiments (other then those mentioned in §4.2) are needed.

The table in fig. 2 outlines the requirements of each version of the simulator with respect to the main functionalities.

3.3 Timeline

Given the software and milestones structures described above, the implementation timeline for the following weeks looks as shown in fig. 3 and fig. 4[8].

The two figures do not present any information about work prior to this Interim Report, as this was mainly literature review on the topics mentioned in §2.
Another aspect not present in the implementation timeline is a schedule of future meetings with the supervisor. Those will be organized regularly as the development proceeds, when milestones are reached or when issues that deserve a discussion arise.

I have planned my work so that I should have the fully working simulator[9] 7 weeks before the Final Report deadline. This is to ensure I have enough time to run experiments, analyze the results, and modify or add functionalities to the simulator in case it is needed.

The system, as described so far, can have some extensions. For example including federated learning enhancements such as client selection (§2.2.A), data selection (§2.2.C), or `SCAFFOLD` (§2.2.D), can influence the performance of the systems. The feasibility and effectiveness of these extensions will be considered later in the development, in case the milestones outlined above are reached sooner than expected.

[8]The numbers enclosed in parenthesis next to some of the entries represent the associated milestone.
[9]This means to reach the 4th milestone, *Learning Methods*.

WEEK	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
STARTING DATE	27/1/20	3/2/20	10/2/20	17/2/20	24/2/20	2/3/20	9/3/20	16/3/20	23/3/20	30/3/20
FYP DEADLINE	(Interim Report)									
EXTERNAL DEADLINE			DL CW		ML Img CW			- ML Img Exam - DL CW	ML Fin CW	
MILESTONES										
1 - MVP										
2 - Mobility										
3 - Learning Tasks										
4 - Learning Methods										
5 - Experiments										
DOCUMENTATION										
Abstract										
Report Draft										
Final Report										
Presentation										
CONFIGURATION										
Basic config file generator (1)										
Basic config file loader (1)										
Configure client mobility (2)										
Selection of learning task (3)										
Selection of learning algorithm (4)										
TRAINING										
Client-level minibatch SGD (1)										
Hierarchical Federated Averaging (1)										
Large mibatch SGD (4)										
Local SGD (4)										
Post-Local SGD (4)										
MODELS										
CNN - MNIST (1)										
ResNet - CIFAR10 (3)										
LSTM - Next-word prediction (3)										
TESTING										
Multilevel model testing on test set (1)										
LOGGING										
Top-down logging infrastructure (1)										
CLIENT MOBILITY										
Add mobility functionality (2)										
TESTING SUITE										
Automated scenario testing (5)										
Simulation results visualizer (5)										

Figure 3: Implementation timeline - first half of the project.

WEEK	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20	Week 21	Week 22
STARTING DATE	6/4/20	13/4/20	20/4/20	27/4/20	4/5/20	11/5/20	18/5/20	25/5/20	1/6/20	8/6/20	15/6/20	22/6/20
FYP DEADLINE									Abstract and Draft		Final Report	Presentation
EXTERNAL DEADLINE				CF Exam		TLDDP Exam						
MILESTONES												
1 - MVP												
2 - Mobility												
3 - Learning Tasks												
4 - Learning Methods												
5 - Experiments												
DOCUMENTATION												
Abstract												
Report Draft												
Final Report												
Presentation												
CONFIGURATION												
Basic config file generator (1)												
Basic config file loader (1)												
Configure client mobility (2)												
Selection of learning task (3)												
Selection of learning algorithm (4)												
TRAINING												
Client-level minibatch SGD (1)												
Hierarchical Federated Averaging (1)												
Large mibatch SGD (4)												
Local SGD (4)												
Post-Local SGD (4)												
MODELS												
CNN - MNIST (1)												
ResNet - CIFAR10 (3)												
LSTM - Next-word prediction (3)												
TESTING												
Multilevel model testing on test set (1)												
LOGGING												
Top-down logging infrastructure (1)												
CLIENT MOBILITY												
Add mobility functionality (2)												
TESTING SUITE												
Automated scenario testing (5)												
Simulation results visualizer (5)												

Figure 4: Implementation timeline - second half of the project.

4 Evaluation Plan

In the following subsections, we will analyze which experiments will be tested and how the simulations results will be compared.

4.1 Key Performance Indicators

Since the main objective of this project is to analyze and optimize learning algorithms in a HFL setting with moving clients, it is necessary to define a set of key performance indicators (KPIs) to be used to evaluate and compare simulations.
The KPIs that we will use are those found in most papers about Federated Learning [1, 9, 12, 13, 15]:

- **Number of learning rounds** needed to achieve a target model accuracy (test for speed of convergence).
- Test **accuracy** reached after N rounds (test for convergence rate).
- **Number and size of messages** sent around in the network.
  This can be coupled with a **latency** analysis of the time needed to send those messages with a modelled communication channel (similar to the analysis done in [1]).

4.2 Outline of Experiments

Simulations, and hence also their outcome, are governed by a set of parameters defined in the configuration file (§3.1.A). To evaluate how different settings influence the performance of the HFL network, the following parameter categories are going to be analyzed:

- Mobility
- Learning task
- Clients learning algorithm
- Servers learning parameters

In **mobility**, the objective is to mimic different real world situations. Three settings will be tested:

- **Urban area**
  High number of devices, the most with high mobility.
- **Rural area**
  Low number of devices, mostly low-mobility (these could be IoT sensors for agriculture).
- **Mixed scenario**
  Mixed environment with high-mobility densely populated clusters (urban area) at the centre, surrounded by low-mobility sparse clusters (rural areas).

For what concerns the **learning task**, we want to benchmark the learning performance on multiple learning tasks:

- **Light model**
  A CNN for the MNIST (or Fashion-MNIST) dataset.
- **Heavy models**
  For example a Resnet18 for CIFAR-10, or a LSTM for next-word prediction based on the Penn Treebank dataset.

In the **clients learning algorithm** category we will evaluate the system behaviour when the client-level learning is performed with various methods, such as minibatch SGD, large minibatch SGD, local SGD, post-local SGD.
Moreover, it is interesting to test the HFL algorithms when the system **servers learning parameters** are varied. These are the global inter-cluster (MBS) update frequency and the intra-cluster (SBS) update frequency.

Note that the parameters categories described above will be cross-tested to look for correlations and behavioural patterns in hierarchical federated learning systems.

This set of experiments will be expanded during the development phase, as the simulator is built. New testing scenarios could also be envisaged if the extension mentioned in §3.3 are implemented.

5 Ethical, Legal, and Safety Issues

The project that has been outlined in the past sections is mainly a research project based on simulations of HFL systems. The data used will be only from publicly available open-source datasets that have been used and are still used by researchers to benchmark their algorithms.
Therefore there is no need for the development of a safety plan and discussion of ethical and legal implications of the outcome of this work.

However, it is interesting to note that, from an ethics perspective, Federated Learning has the great potential of allowing the development of more sophisticated and accurate models, as many more users will agree to participate, thanks to the promise of increased privacy. Moreover, because of this privacy-enhancing aspect, some business areas that were reluctant in sharing their data (e.g. hospitals [16] and the medical field in general) are now exploring this field.

References

[1] Mehdi Salehi Heydar Abad et al. *Hierarchical Federated Learning Across Heterogeneous Cellular Networks*. 2019. arXiv: 1909.02362 [cs.LG].

[2] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: 10.1145/3133956.3133982. URL: https://doi.org/10.1145/3133956.3133982.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] Priya Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2017. arXiv: 1706.02677 [cs.CV].

[5] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[6] Kevin Hsieh et al. *The Non-IID Data Quagmire of Decentralized Machine Learning*. 2019. arXiv: 1910.00189 [cs.LG].

[7] Sergey Ioffe. "Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models". In: *CoRR* abs/1702.03275 (2017). arXiv: 1702.03275. URL: http://arxiv.org/abs/1702.03275.

[8] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[9] Sai Praneeth Karimireddy et al. *SCAFFOLD: Stochastic Controlled Averaging for On-Device Federated Learning*. 2019. arXiv: 1910.06378 [cs.LG].

[10] Tao Lin et al. *Don't Use Large Mini-Batches, Use Local SGD*. 2018. arXiv: 1808.07217 [cs.LG].

[11] Yujun Lin et al. "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training". In: *CoRR* abs/1712.01887 (2017). arXiv: 1712.01887. URL: http://arxiv.org/abs/1712.01887.

[12] Lumin Liu et al. "Edge-Assisted Hierarchical Federated Learning with Non-IID Data". In: *CoRR* abs/1905.06641 (2019). arXiv: 1905.06641. URL: http://arxiv.org/abs/1905.06641.

[13] H. Brendan McMahan et al. "Federated Learning of Deep Networks using Model Averaging". In: *CoRR* abs/1602.05629 (2016). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629.

[14] Kensuke Nakamura and Byung-Woo Hong. "Adaptive Weight Decay for Deep Neural Networks". In: *CoRR* abs/1907.08931 (2019). arXiv: 1907.08931. URL: http://arxiv.org/abs/1907.08931.

[15] Takayuki Nishio and Ryo Yonetani. "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)* (May 2019). DOI: 10.1109/icc.2019.8761315. URL: http://dx.doi.org/10.1109/ICC.2019.8761315.

[16] Pulkit Sharma, Farah E Shamout, and David A Clifton. *Preserving Patient Privacy while Training a Predictive Model of In-hospital Mortality*. 2019. arXiv: 1912.00354 [cs.LG].

[17] Shaohuai Shi et al. "A Distributed Synchronous SGD Algorithm with Global Top-k Sparsification for Low Bandwidth Networks". In: *CoRR* abs/1901.04359 (2019). arXiv: 1901.04359. URL: http://arxiv.org/abs/1901.04359.

[18] Tiffany Tuor et al. *Data Selection for Federated Learning with Relevant and Irrelevant Data at Clients*. 2020. arXiv: 2001.08300 [cs.LG].