Understanding of Federated Learning:

Machine Learning Model and Dataset:

For this project, we have used a Linear Regression machine learning model to predict housing values based on the California Housing Dataset. The dataset is made up of 20640 total data samples and consists of 8 features including the median income, housing median age, average rooms, average bedrooms, population, average occupancy, and geographical factors like latitude and longitude. There is also one target variable, the median house value. The features included in the dataset are common potential predictors of house prices and thus enable our model to reliably predict the median house prices based on the physical properties and location of houses. Due to the presence of a single target variable, and our objective being a regression problem, the Linear Regression model is suitable choice for predicting the house value based on the given 8 regression factors. This suggests that there would be a linear correlation between the target variable and at least one regression factor. Since our scenario has 5 clients, the dataset is split up across each client, with each client having their own unique testing and training data on which they train their local model.

Federated Learning Algorithm:

The Federated Learning algorithm implemented in our project is the FedAvg (Federated Averaging) algorithm. The FedAvg algorithm is a core approach in Federated Learning which allows for collaborative model training across multiple clients while keeping each client's data private. Our system consists of 1 server and 5 clients where each client has access to its own private dataset which is a unique subset of the California Housing Dataset. In the FedAvg algorithm, each client independently trains a local model on its data, and then sends the model updates, rather than the actual data, to the server. The server then aggregates the updates received from the clients to improve the current global model, and then distributes the new global model back to the clients so they can continue local training and updating. By decentralising data, it maintains data security and privacy, and reduces the storage strain on the server as the data is stored on clients instead which in turn improves scalability by simply adding on clients to the system.

Implementation:

To implement the program, we have two python programs. One handles the server side and the other the client. The server first waits for a client to connect, sending a confirmation message when it receives its handshake, then waits 30 seconds before starting the Federated Learning. This is done by sending out the global model to all connected clients, then waiting for them to send their local models back. Once the local models have been received, the server performs the FedAvg algorithm as seen in Algorithm 1. If a client attempts to join it gets added at the start of the next iteration. This is done for 100 iterations.

The client immediately sends a handshake to the server, until it receives a confirmation. Once it receives the global model, it performs the Linear Regression model using a Gradient Descent algorithm as seen in Algorithm 2. Where necessary we performed Mini-Batching as seen in Algorithm 3 before Gradient Descent. The client then continuously attempts to send the local model to the server until it receives a confirmation.

Experimental Results:

Simulation Results:

The simulation results produced by our system demonstrate the significant improvements made to the global model over the course of multiple iterations. Both the testing and training mean squared errors (MSE) showed major reductions from the first iteration to the last iteration, gradually decreasing over the course of the program. In Figure 1, we can see the same trend across each client's data where there is a rapid decline in the MSE over the initial iterations followed by a more gradual decline as the MSE nears 0.

By examining the logs for each client, we can see the final iteration produces a substantially lower testing MSE and training MSE. For example, the initial test MSE on client 1's data was over 309,000 and rapidly decreased to 196.77 after a single iteration. Then by the 10th iteration it was reduced to 4.18, and by the 100th iteration it was only 2.47. Similarly, the other clients demonstrate the same trend but with slightly different values. Client 2's testing MSE reduced from 367,000 to 2.45, client 3 from 389,000 to 2.78, client 4 from 389,000 to 2.55, and client 5 from 326,000 to 2.41 (see Figure 2).

In addition, the pre-update and post-update training MSE results also follows a downwards trend. In Figure 3, we can see client 1's results in its first 4 iterations with the first iteration holding a pre-update training MSE of 402,700 which drastically drops to 223.06 after the update. The training MSE continues to be reduced in each subsequent iteration, with the fourth iteration showing that the pre-update training MSE is now only 55.43 which is significantly less than in the first iteration, and this is reduced even further after the update for a post-update training MSE of 24.85.

The convergence of testing and training MSE highlight the system's success in learning from the datasets. Given that our simulation results demonstrate a constant decrease in the testing MSE after each iteration, we can conclude that the server is successfully working to improve the global model. Likewise, the downwards trend in the training MSE results for every client indicates that the local model is also successfully learning and improving alongside the global model. As each client improves its local model, and as the server also improves its global model, the iterative broadcasting of updates allows for all the models to continue improving together as expected by a Federated Learning algorithm.

Accomplished and Remaining Tasks:

We have completed the essential criteria such as the integration of the FedAvg algorithm. Its success in learning and refining the existing models continuously is evident as per the simulation results above which outline the MSE trends that we anticipated by a functional FedAvg system. We also implemented two separate optimisation methods: gradient descent and mini-batch gradient descendent. In conjunction, there is an option for enabling subsampling and selecting the number of sub-clients to randomly select in each iteration.

The essential criteria have been satisfied. However, there are some potential tasks that remain unsolved and could improve our understanding of federated learning. This includes implementing other federated learning algorithms such as FedSGD or FedDyn which we could have compared with the results of FedAvg and performed an analysis to determine the pros and cons of each approach.

Comparative Analysis:

Gradient Descent vs Mini-Batch GD:

Gradient Descent (GD) updates the model using the entire training dataset to compute the gradient of the loss function. It is more stable and consistent in reaching convergence since it considers the entire training dataset. However, this is computationally expensive since all the data is processed in a single batch and takes longer to reach convergence. After experimenting with different values, we optimised our model by setting the number of epochs to 300 and the learning rate to 1e-7 because other rates would often cause NaN and Inf MSE scores.

Mini-Batch Gradient Descent (MBGD) divides the available training data into smaller subsets or batches that are used to calculate the MSE and update the model coefficients. This leads to convergence in less iterations than GD as the model is updated more frequently, however, each iteration takes longer to compute since multiple loss functions are performed due to multiple batches. For our parameters, we set the number of epochs to 100 as convergence is reached faster and iteration times take longer than in GD. We chose a batch size of 64 because size 32 took longer to compute and size 128 resulted in lower accuracy and higher MSE, making 64 a good balance. Our learning rate remained the same as in GD at 1e-7 as increasing or decreasing led to NaN and Inf MSE scores.

Figure 1 and Figure 4 display the MSE over iterations for GD, while Figure 5 and Figure 6 show the MSE over iterations for MBGD. It is evident that given that both optimisation methods have their optimal parameters set, MBGD results in significantly quicker convergence than standard GD. Although MBGD had a higher individual iteration time, the need for less epochs makes up for it, and it also provides less computational strain as the entire dataset is not processed at once.

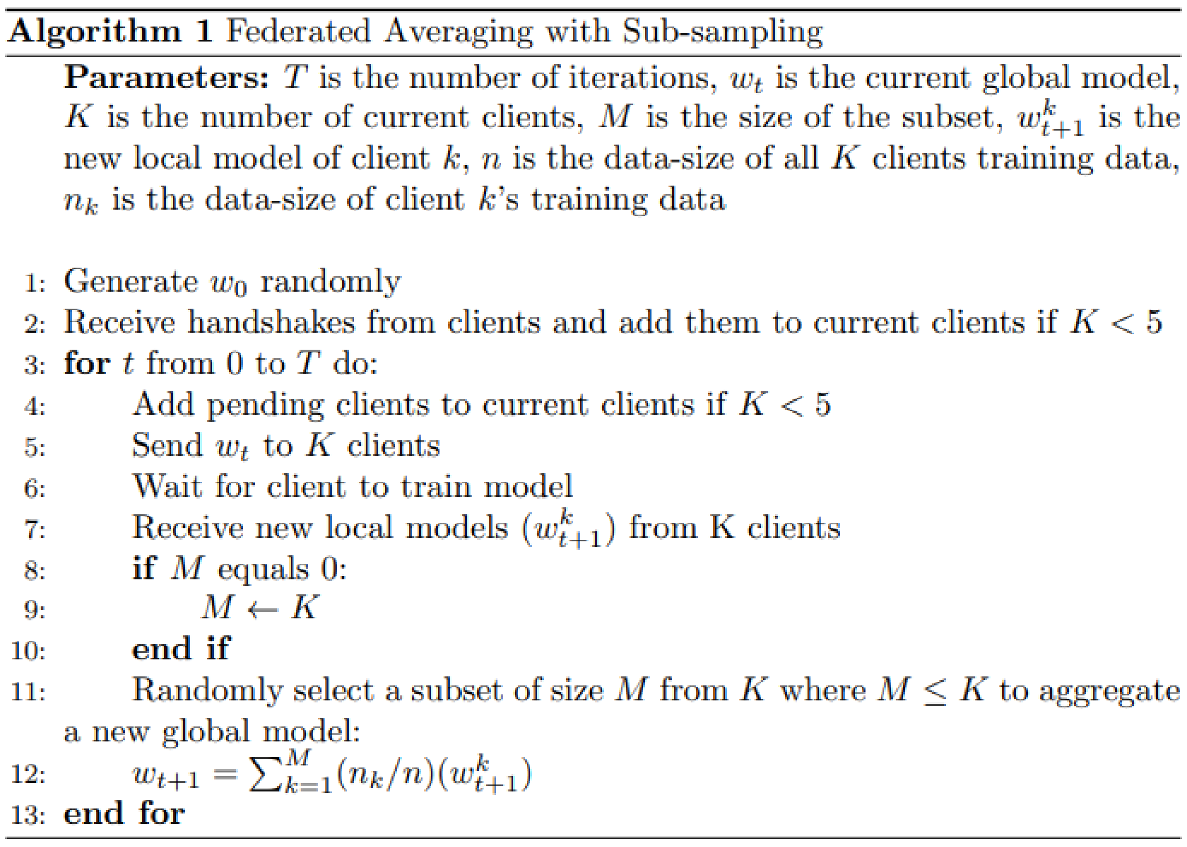
Subsampling:

Subsampling is the process where, in each iteration, a randomly selected subset of clients receives the global model, locally train and update their model, and sends the updates back to the server for aggregation to improve the global model. This differs to non-subsampling which includes all clients every iteration. Subsampling is more computationally performant and scalable since fewer clients are involved. In addition, the random selection of clients prevents overfitting the global model to a particular data subset. However, the MSE may be less stable and present bias since there is no guaranteed fairness or consistency in the data used.

Every iteration of the program, the server will randomly select a subset of clients to use for improving the local and global models during that iteration. In our program, this number is set in the command-line arguments when launching the server. The server then distributes its global model to the sub-clients, who will then train their local model given the new global model and their own private data. Only the sub-clients will return their updates back to the server, where the server aggregates the updates and improves the existing global model. In the next iteration, the process will repeat where another randomly selected subset of clients is chosen.

As seen in Figure 7 and Figure 8, each client has only plotted points when the client was selected in that iteration, with 2 sub-clients per round. Subsampling seems to yield lower MSE in this case, but this is not consistent across multiple runs due to the data sets being randomly selected. The primary advantage of subsampling comes from its faster computational speed and reduced communication overhead compared to non-subsampling due to less sub-clients.

Appendix:



Algorithm 1 Federated Averaging with Sub-sampling

Parameters: T is the number of iterations, w_t is the current global model, K is the number of current clients, M is the size of the subset, w_{t+1}^k is the new local model of client k, n is the data-size of all K clients training data, n_k is the data-size of client k's training data

```
1:  Generate $w_0$ randomly
2:  Receive handshakes from clients and add them to current clients if $K < 5$
3:  for $t$ from 0 to $T$ do:
4:      Add pending clients to current clients if $K < 5$
5:      Send $w_t$ to $K$ clients
6:      Wait for client to train model
7:      Receive new local models $(w_{t+1}^k)$ from K clients
8:      if $M$ equals 0:
9:          $M \leftarrow K$
10:     end if
11:     Randomly select a subset of size $M$ from $K$ where $M \leq K$ to aggregate
    a new global model:
12:     $w_{t+1} = \sum_{k=1}^{M} (n_k/n)(w_{t+1}^k)$
13: end for
```

Algorithm 1: Federated Average with Subsampling Algorithm

Algorithm 2 Gradient Descent

Parameters: E is the number of local epochs, $\hat{Y}$ is the predicted dependent variables using the current model, M is the set of gradients, c is the bias, MSE is the mean squared error, n is the data-size of the clients training data, Y is the set of dependent variables in the training data, X is the set of independent variables in the training data, D_m is the partial derivative with respect to m, D_c is the partial derivative with respect to m, η is learning rate

```
for $e$ from 0 to $E$ do:
    $\hat{Y}_i = MX + c$
    $MSE = 1/n \sum_{i=0}^{n} (Y_i - \hat{Y}_i)^2$
    $D_M = -2/n \sum_{i=0}^{n} X_i(Y_i - \hat{Y}_i)$
    $D_c = -2/n \sum_{i=0}^{n} (Y_i - \hat{Y}_i)$
    $M = M - \eta D_M$
    $c = c - \eta D_c$
end for
```

Algorithm 2: Gradient Descent Algorithm

Algorithm 3 Mini-Batching

Parameters: E is the number of local epochs, T is the training data set, B is the batch-size

```
for $e$ from 0 to $E$ do:
    Randomly divide T into batches of B size
    for $T_b$ in $T$ do:
        Perform one epoch of Gradient Descent on $T_b$
    end for
end for
```

Algorithm 3: Mini-Batching Algorithm

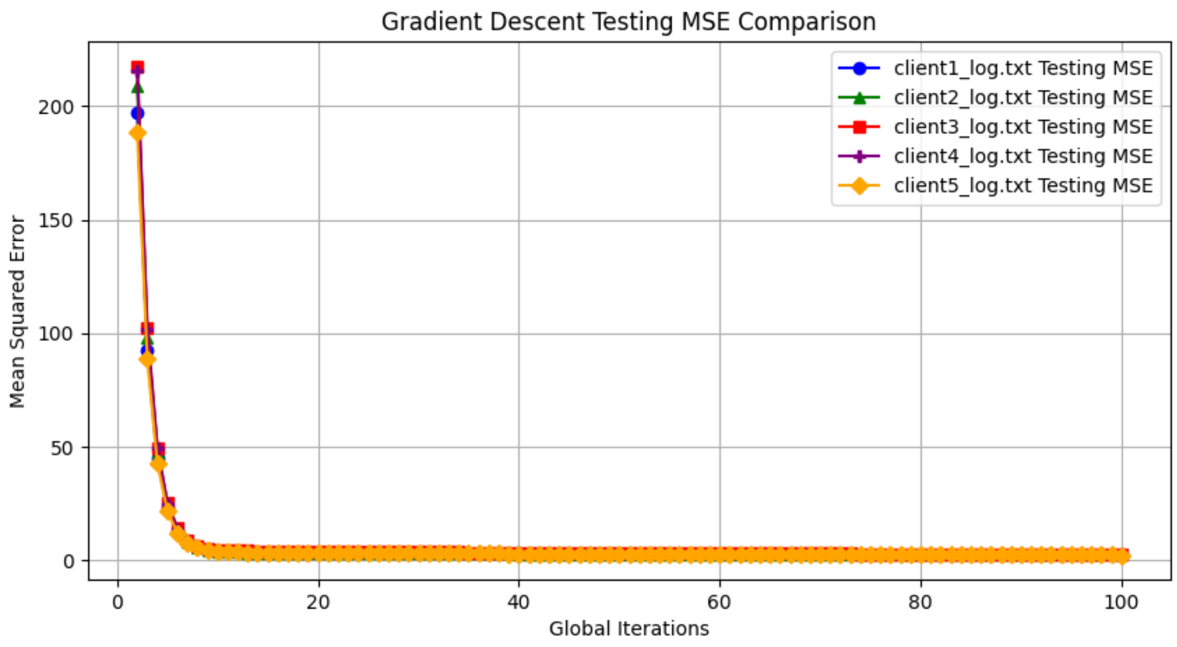
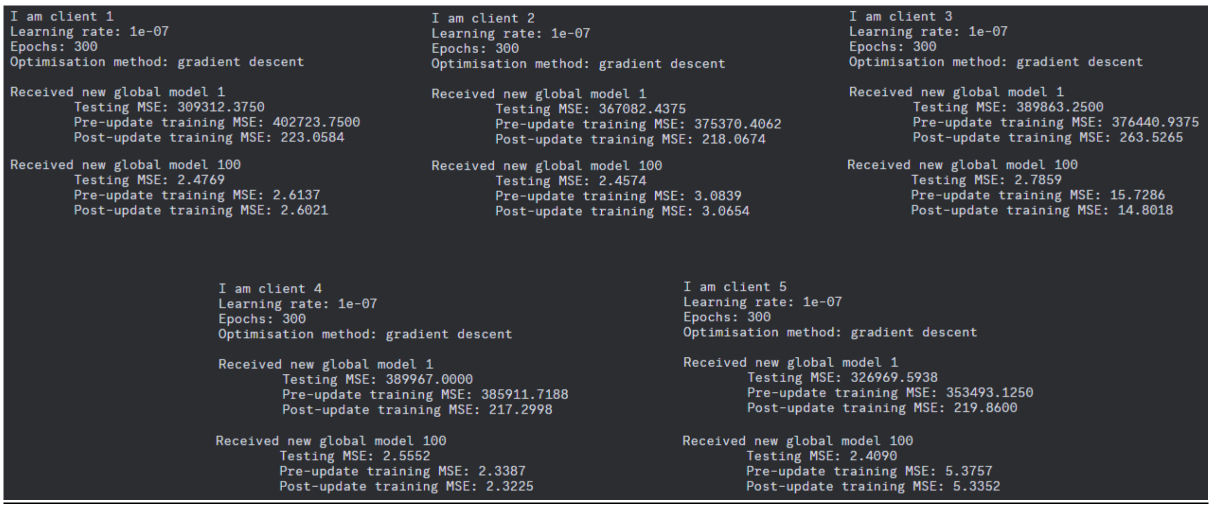


Figure 1: Testing MSE across clients using Gradient Descent (excluding initial testing MSE due to being large outlier)



```
I am client 1
Learning rate: 1e-07
Epochs: 300
Optimisation method: gradient descent

Received new global model 1
        Testing MSE: 309312.3750
        Pre-update training MSE: 402723.7500
        Post-update training MSE: 223.0584

Received new global model 100
        Testing MSE: 2.4769
        Pre-update training MSE: 2.6137
        Post-update training MSE: 2.6021
```

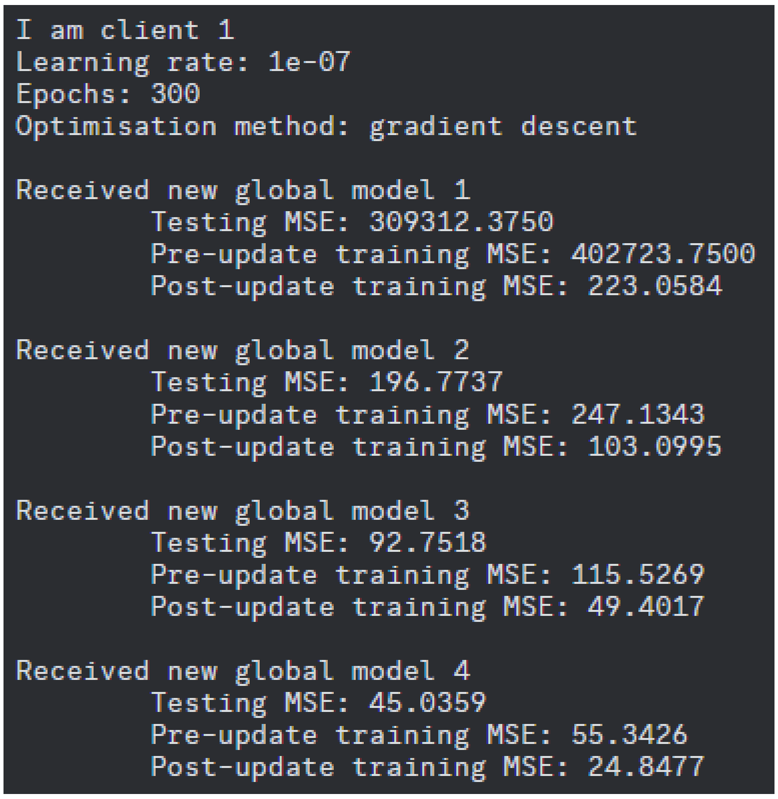
```
I am client 2
Learning rate: 1e-07
Epochs: 300
Optimisation method: gradient descent

Received new global model 1
        Testing MSE: 367082.4375
        Pre-update training MSE: 375370.4062
        Post-update training MSE: 218.0674

Received new global model 100
        Testing MSE: 2.4574
        Pre-update training MSE: 3.0839
        Post-update training MSE: 3.0654
```

```
I am client 3
Learning rate: 1e-07
Epochs: 300
Optimisation method: gradient descent

Received new global model 1
        Testing MSE: 389863.2500
        Pre-update training MSE: 376440.9375
        Post-update training MSE: 263.5265

Received new global model 100
        Testing MSE: 2.7859
        Pre-update training MSE: 15.7286
        Post-update training MSE: 14.8018
```

```
I am client 4
Learning rate: 1e-07
Epochs: 300
Optimisation method: gradient descent

Received new global model 1
        Testing MSE: 389967.0000
        Pre-update training MSE: 385911.7188
        Post-update training MSE: 217.2998

Received new global model 100
        Testing MSE: 2.5552
        Pre-update training MSE: 2.3387
        Post-update training MSE: 2.3225
```

```
I am client 5
Learning rate: 1e-07
Epochs: 300
Optimisation method: gradient descent

Received new global model 1
        Testing MSE: 326969.5938
        Pre-update training MSE: 353493.1250
        Post-update training MSE: 219.8600

Received new global model 100
        Testing MSE: 2.4090
        Pre-update training MSE: 5.3757
        Post-update training MSE: 5.3352
```

Figure 2: Logs of the exact initial and final MSE results for each client



```
I am client 1
Learning rate: 1e-07
Epochs: 300
Optimisation method: gradient descent

Received new global model 1
        Testing MSE: 309312.3750
        Pre-update training MSE: 402723.7500
        Post-update training MSE: 223.0584

Received new global model 2
        Testing MSE: 196.7737
        Pre-update training MSE: 247.1343
        Post-update training MSE: 103.0995

Received new global model 3
        Testing MSE: 92.7518
        Pre-update training MSE: 115.5269
        Post-update training MSE: 49.4017

Received new global model 4
        Testing MSE: 45.0359
        Pre-update training MSE: 55.3426
        Post-update training MSE: 24.8477
```

Figure 3: Logs of the gradual decrease in MSE over first four iterations for client1

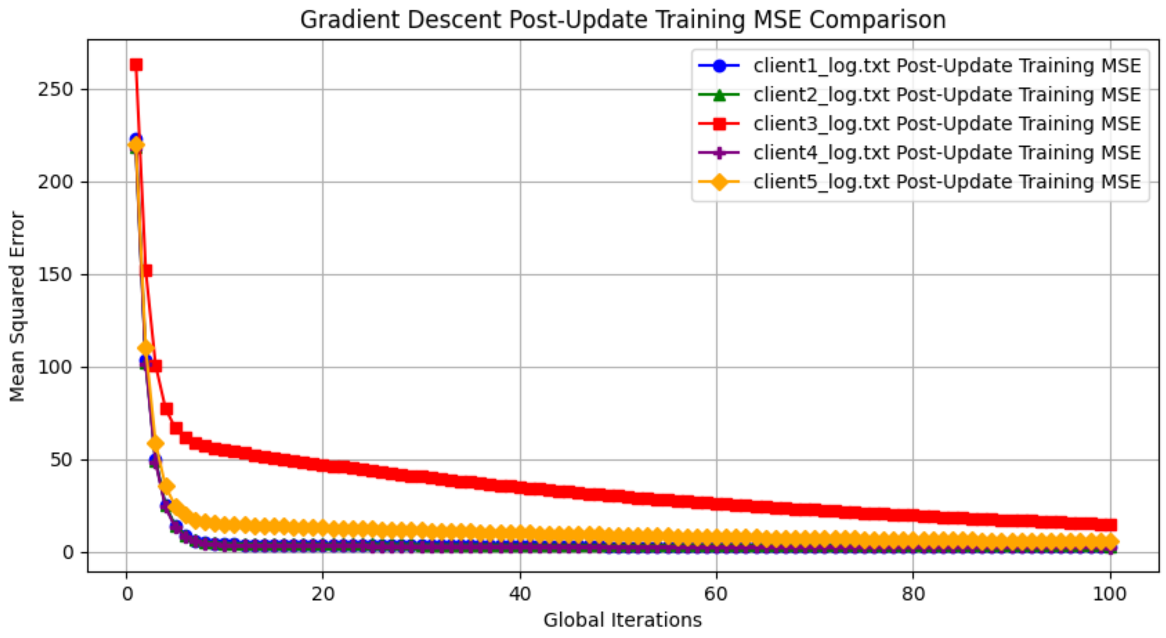


Figure 4: Post-update training MSE across clients using Gradient Descent

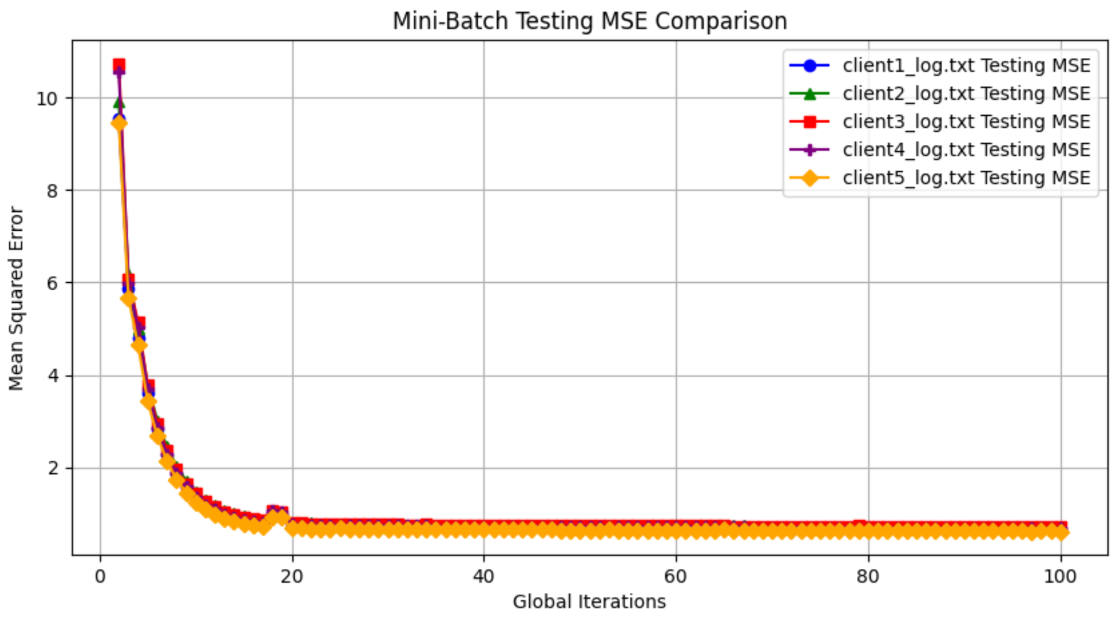


Figure 5: Testing MSE across clients using Mini-Batch GD

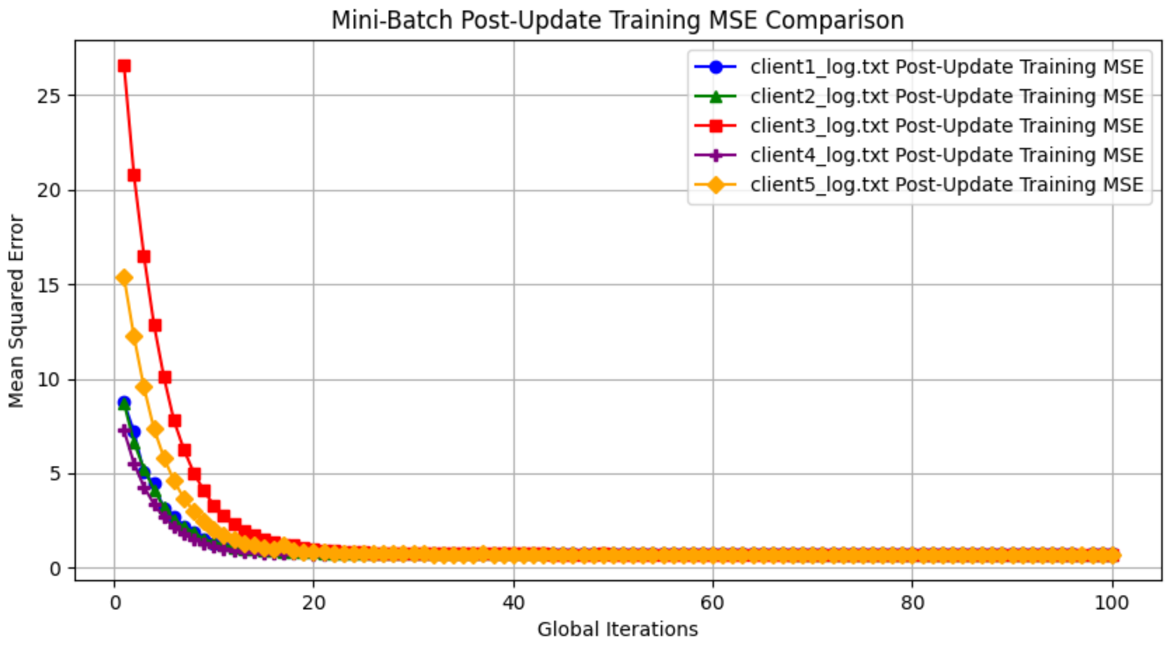


Figure 6: Post-update training MSE across clients using Mini-Batch GD

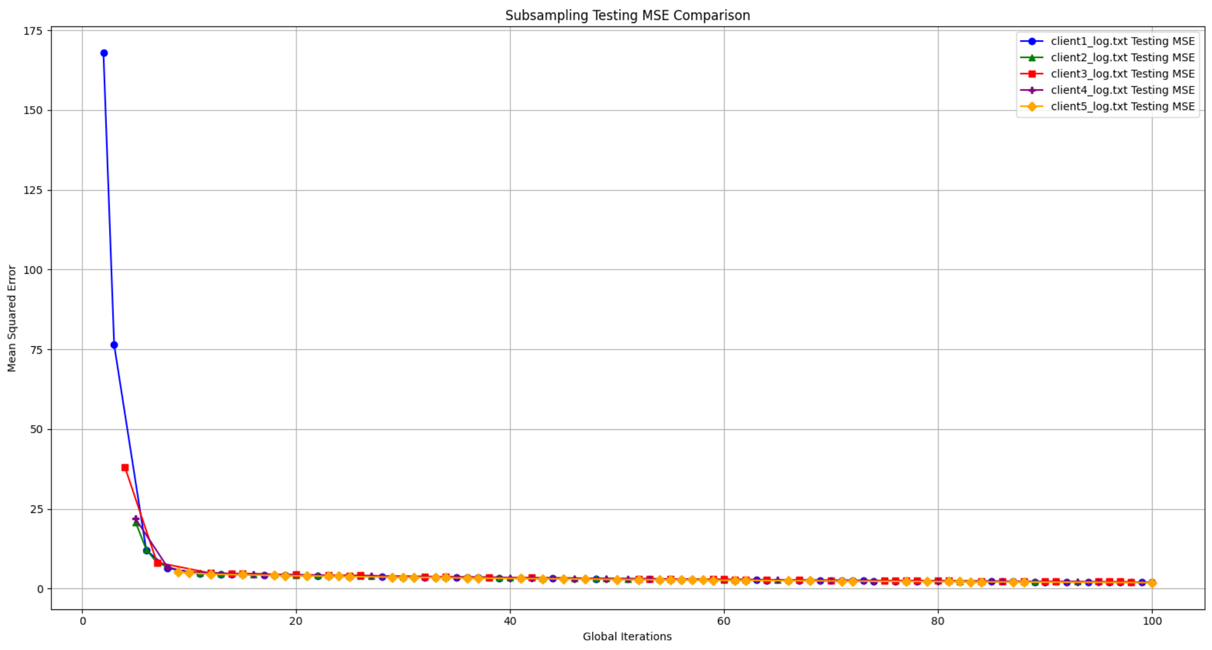


Figure 7: Testing MSE across clients using size 2 subsampling Gradient Descent

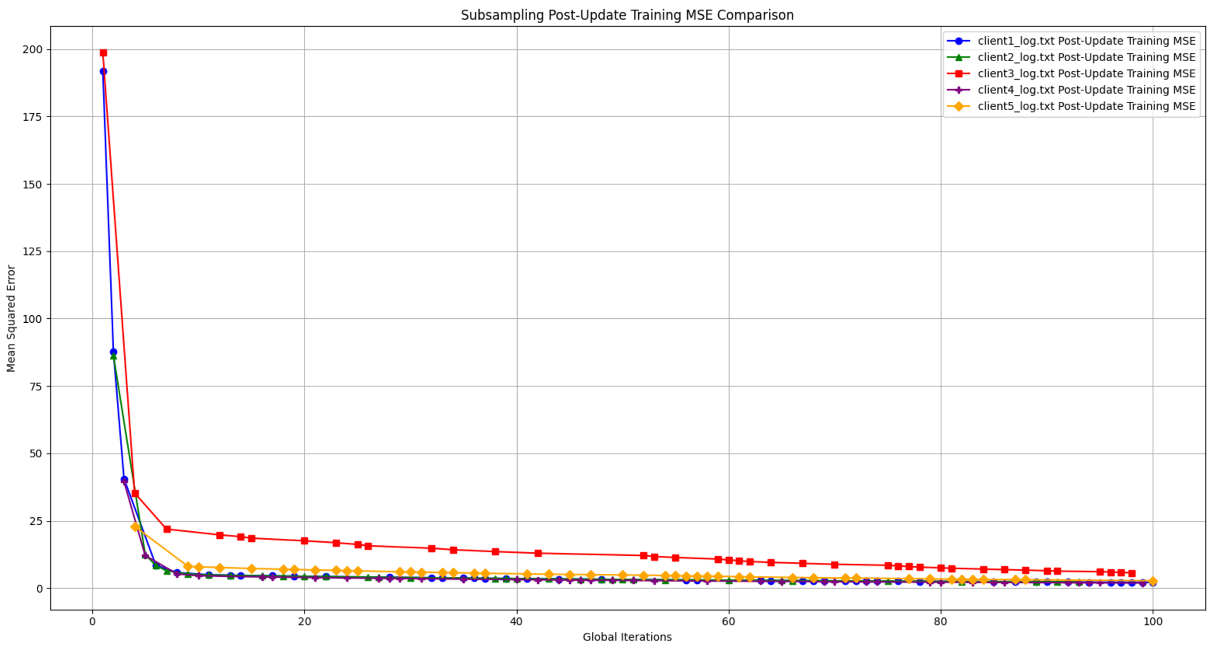


Figure 8: Post-update training MSE across clients using size 2 subsampling Gradient Descent