# FedDC Experiments

**Jay Lindsay**
University of Cambridge
Fitzwilliam College
jl2272@cam.ac.uk

## Abstract

Federated Daisy Chaining (FedDC) is a new method of federated learning that aims to improve performance when clients have a small number of samples. This report describes how the strategy was implemented into the Flower framework, and attempts to reproduce its performance on experiments similar to those done within the paper that proposed it. New experiments are also done to challenge FedDC, and a new method of aggregation is implemented and tested to see if it can make improvements in less favourable scenarios. Additional analysis is carried out to compare convergence times against other strategies. The experiments find that FedDC performs very well in all scenarios tested, however training may take longer than with other methods.

## 1 Introduction

Traditional federated learning suffers when clients have a small amount of data. The quality of the resulting locally trained models can be reduced, which leads to a lower-quality aggregated model. The proposed method by Kamp et al. [2023] aims to solve this via a "daisy chaining" strategy. Rather than aggregating the client-produced parameters each round, the server instead redistributes these parameters to other clients which forms a "daisy chain". The clients then train on another client's parameters instead of aggregated ones. After an "aggregation period" number of rounds has passed, we aggregate and distribute the parameters as normal. This method has shown promising results on scenarios with a limited amount of data per client.

This report reproduces the effectiveness of this strategy empirically by comparing the approach against FedAvg, FedAdagrad, FedYogi, FedAdam and centralised training on various subsets and partitions of the MNIST dataset. The strategy is tested on two scenarios similar to those in the original paper, these being an identical small number of IID and non-IID samples on each client. This allows us to verify the paper's findings. In addition, we consider a previously untested scenario with both label distribution and quantity skew. While the paper primarily focuses on the final results, I will also be discussing the practicality of the method in terms of time taken to converge. An alternative method of aggregation is also proposed and tested, that being a weighted average by the total number of samples a set of parameters has been trained on. The data produced shows that FedDC does indeed outperform or match the methods compared against it, even approaching the centralised gold standard. However, training can take significantly longer in some scenarios.

## 2 Background

This section consists of a detailed description of the FedDC algorithm and existing code as produced by Kamp et al. [2023], a description of the experiments performed already, and an analysis of these experiments (including a justification of why more experiments are needed).

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

### 2.1 FedDC

FedDC (Algorithm 1) requires two additional input parameters. The daisy-chaining period $d$ refers to the number of local training rounds before sending our model to the server for daisy-chaining, while the aggregation period $b$ refers to the number of local training rounds before sending for aggregation. An aggregation round takes priority over a daisy-chaining round.

**Algorithm 1** Federated Daisy-Chaining FedDC (Kamp et al. [2023])

**Input:** daisy-chaining period $d$, aggregation period $b$, learning algorithm $\mathcal{A}$, aggregation operator $agg$, $m$ clients with local datasets $D^1, \ldots, D^m$, total number of rounds $T$
**Output:** final model aggregate $h_T$

initialize local models $h_0^1, \ldots, h_0^m$
**Locally at client** $i$ **at time** $t$ **do**
  sample $S$ from $D^i$
  $h_t^i \leftarrow \mathcal{A}(S, h_{t-1}^i)$
  **if** $t \mod d = d - 1$ **or** $t \mod b = b - 1$ **then**
    send $h_t^i$ to server
    receive new $h_t^i$ from server
**At server at time** $t$ **do**
  **if** $t \mod d = d - 1$ **then**
    draw permutation $\pi$ of $[1, m]$ at random
    **for all** $i$ **in** $[m]$ **send model** $h_t^i$ to client $\pi(i)$
  **else if** $t \mod b = b - 1$ **then**
    $h_t \leftarrow agg(h_t^1, \ldots, h_t^m)$
    send $h_t$ to all clients

As described in the introduction, the algorithm works by requiring each client to send their parameters to a random other client (via the server) on a daisy-chaining round. The server only performs aggregation on an aggregation round.

It is important to note that there appears to be an error within this pseudocode. The pseudocode checks for a daisy-chaining round before an aggregation round, when in reality these should be reversed (or we would be stuck daisy-chaining forever with d=1). The source code and previous experimental results confirm this to be an error.

### 2.2 Previous experiments and existing code

Previous experiments have been done on many datasets including CIFAR10, SUSY, two real medical datasets and MNIST all with small sample sizes per client. In these experiments, each client had an identical number of samples (with one exception being a simulated realistic client distribution of MRI scans). FedDC was compared against FedAvg, FedProx, FedAdagrad, FedYogi, FedAdam and centralised learning. The paper's proposed use case for FedDC is a cross-silo scenario with limited amounts of data.

Each experiment found that FedDC outperformed all other types of learning, with the exception being centralised learning.

The source code used to perform these experiments is available. This code uses PyTorch and simulates federated learning by storing a list of clients and averaging/distributing the parameters as needed.

### 2.3 Strengths and weaknesses of existing work

Many experiments have already been performed, providing a lot of empirical evidence for FedDC's effectiveness. Each experiment was repeated 3 times to reduce the likelihood of anomalies. However, the scenarios were all quite similar in scope. There is only one scenario in which all clients did not have the same number of samples, and these samples were IID. Clients with varying qualities of data were not considered. In addition, only the final accuracies were reported in the tables, with quite a high number of rounds being used to train the models. This suggests that FedDC may need a long time to converge but this is not discussed. Finally, the source code provided is very inflexible since

it is hard-coded to work for the datasets provided, meaning the strategy cannot currently be easily tested with other datasets.

Performing further experiments will help to verify the paper's findings and provide more evidence that FedDC is a viable strategy, in addition to verifying that the strategy does not completely fail in other scenarios. We can also determine if there is any negative impact on convergence times. Implementing it into Flower will solve the issues with inflexibility and allow further research to be done on the strategy.

## 3 Design/Methods

### 3.1 FedDC refactored pseudocode

Within the pseudocode described in the paper (Algorithm 1), the rounds referred to are local client training rounds (or local epochs). The Flower framework considers a round to be each communication between client and server, hence to implement this into the Flower framework we must refactor the algorithm.

**Algorithm 2** FedDC Refactored

**Input:** daisy-chaining period $d$, aggregation period $b$ such that $b$ mod d = 0, learning algorithm $\mathcal{A}$, aggregation operator agg, $m$ clients with local datasets $D^1, \ldots, D^m$, total number of rounds $T$
**Output:** final model weights $w_T$

**Server executes:**
  initialise $w_0$
  **for** $t = 1$ **to** $T$ **do**
    draw permutation $\pi$ of $[1, m]$ at random
    **for** $k = 1$ **to** $m$ **in parallel do**
      **if** $t \mod (b/d) = 0$ **or** $t = 1$ **then**
        $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t)$
      **else**
        $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t^{\pi(k)})$
      **end if**
    **end for**
    $w_{t+1} \leftarrow \text{agg}(w_t^1, \ldots, w_t^m)$
  **end for**

**procedure** CLIENTUPDATE$(k, w)$
    **for** $i = 1$ **to** $d$ **do**
      sample $S$ from $D^k$
      $w \leftarrow \mathcal{A}(S, w)$
    **end for**
    **return** $w$
**end procedure**

There are a few minor changes in functionality. Firstly, the initial weights are initialised server-side and distributed to the clients in the first round, which means different clients cannot have different initial parameters. This does not have much of an impact. In addition, we aggregate at every round even if we don't send this to clients, this will help with evaluation. We only consider cases where the aggregation period is a multiple of the daisy-chaining period; this decision was made to keep the amount of training done at each client constant between server communication rounds. This will somewhat limit the possible experiments we can perform, however, I believe this simplification makes the strategy far easier to understand and perform further research on without having a huge impact (all experiments within the paper, with the exception of one, use a multiple of the daisy-chaining period for the aggregation period). The final change is that the rounds are now server communication rounds, and hence the original input parameter $T$ will need to be changed accordingly by dividing by $d$.

### 3.2 Flower implementation

As the paper suggests averaging to be used as the aggregation operator for deep learning, the FedAvg strategy is used as a base for the FedDC strategy. Weighted averaging does not make sense in this context since the number of samples the parameters are trained on before aggregation is not equal to the number of samples of the final client. Hence, we use unweighted averaging.

As seen from the refactored pseudocode above, the daisy-chaining period is identical to the local epoch number, and hence we can alter this via modification to this parameter within the client. Therefore, we only need to pass in one parameter to the Flower strategy (b/d), we call this the aggregation_period for simplicity. The rest of the parameters are the same as FedAvg.

Both the configure_fit and aggregate_fit methods are overridden and modified. We get access to each client's parameters within the aggregate_fit method and store these to be used when configuring the next round. When configuring a training round, we check if we are currently on an aggregation round. If we are, we send each client the aggregated parameters. Otherwise, we are on a daisy-chaining round and can send each client the parameters stored at the end of the previous aggregation round. The random permutation is handled by randomly sampling clients while configuring training.

### 3.3 Alternate aggregation operator

While unweighted averaging was used in previous experiments with FedDC, this may not be the best choice. FedAvg performs a weighted average by the number of samples per client, and this has been found to improve performance. We could do something similar by counting the number of samples a set of parameters has been trained on before aggregation, this is done by totaling the samples of each client in the daisy chain. I will refer to this method as FedDCW and will experiment with this new strategy when there are a varying number of samples per client to see if performance is improved.

## 4 Experimental setup and methodology

### 4.1 Experiments on an IID tiny subset of MNIST

The goal of this experiment is to reproduce the findings from the paper that FedDC outperforms other strategies when we have a small amount of IID data per client. Prior to evaluation, I hypothesise that this experiment will produce similar results to what was found in the paper due to the vast amount of empirical evidence for this scenario.

To produce this dataset, we first need to download the full MNIST dataset. We produce the partition at run-time since storing partitions uses a significant amount of storage space. We create the dataset by taking (Number of clients $\times$ Number of samples) samples from the full MNIST train set. This is seeded to ensure each experiment receives the same dataset. We then assign these samples to the clients. 8 samples are assigned to 25 clients, meaning our training subset is only 0.33% of the total available train set. This is intentional, as it ensures we don't easily get a very high accuracy for every strategy which would make them hard to compare.

### 4.2 Experiments on a non-IID tiny subset of MNIST

The main goal of this experiment is to test FedDC under data heterogeneity, in particular label distribution skew. Similar experiments have been done by the paper, so we are once again also testing for reproducibility. As there is also evidence to support that FedDC can perform well under these conditions, my hypothesis is that we will again see strong performance.

As before, we create the partitions at run-time using a seed. The train set is sorted by the labels to ensure every small subarray contains almost the same labels. We assign a random subarray to each client that is equivalent to the sample size, in this case, we have 8 samples in 25 clients again.

### 4.3 Experiments with both quantity skew and label distribution skew

The main goal of this experiment is to challenge FedDC in a worst-case scenario. We create a scenario in which 80% of our clients have a low number of samples with very similar or identical labels (homogenous data within a client, but heterogenous to other clients). The other 20% are clients with

a large amount of data that matches the global distribution. This scenario could arise in reality if we train with both large institutions that would own large amounts of varied data and standard users who would have small amounts of more homogeneous data. My hypothesis is that FedDC will perform extremely poorly in this scenario as each set of parameters will be almost guaranteed to be trained on the "bad" small clients before aggregation which would decrease the quality of the model. However, I believe that FedDCW could perform better as the weighted averaging could move the aggregated parameters toward the data contained within the large clients.

The dataset is created using a combination of the previous two approaches. We generate 20 small clients each with 8 similar samples by sorting the train set and taking out subarrays. We then generate 5 large clients by randomly sampling 1200 samples from the remaining train data.

### 4.4 Strategies used

For each experiment, we compare FedDC with $b$=5, $b$=10 and $d$=1 against FedAdam, FedAdagrad, FedAvg, FedYogi and centralised training on the combined data from every client. A daisy-chaining period/epoch number of 1 was used in federated learning to reduce training times. For FedAdam, FedAdagrad and FedYogi, the parameters used are the default present within the CaMLSys FL Project Template as these work well on the MNIST dataset.

250 rounds of training were performed for the identical sample experiments as accuracy continually improved up to this point in some scenarios (with the exception being centralised training as this converged very quickly, here only 50 rounds were used). Only 100 rounds were used for the quantity skew experiment, as accuracies converged much quicker. We again use 50 rounds for the centralised case. FedDCW was also tested in this experiment with $b \in \{5, 10\}$ to see if my proposed improvement could affect performance when we have quantity skew. Other experiments would not be affected due to each client having the same sample size.

While $b$=10 seemed to work the best within the experiments done within the paper, $b$=5 was also tried because the number of rounds used in this report's experiments was far lower. The idea was that aggregation might be performed too infrequently with $b$=10.

Comparing against centralised training is important as it represents the "gold standard" of what could possibly be achieved. Obtaining an accuracy close to this would be ideal.

### 4.5 Evaluation

The primary metric measured is test accuracy on the centralised MNIST test set. The aggregated parameters from every round will be tested on this test set. With FedDC, we aggregate the parameters every round even if they are not used so we can still test before an actual aggregation round. Tables for final accuracies will be produced, along with graphs to show the progression of the average test accuracy over rounds. This should give a good indication of both the quality of the final model and the performance of the strategy in terms of rounds taken to converge.

### 4.6 Minimising error

Every experiment is repeated three times to reduce the risk of an erroneous conclusion. We consider a strategy to outperform another when the maximum achieved by one strategy is lower than the minimum from another. This should reduce the risk of errors, although since the number of experiments is low due to compute constraints it will still be possible for errors to happen. A different seed is used for client selection on each run, and training is done completely randomly. The only seed maintained between runs is the dataset partitioning seed to ensure we are testing on the same data every time. A median, min and max for each experiment will be reported. Details on how to reproduce the experiments are included within the appendix.

### 4.7 Model used and client information

Each experiment will train a Convolutional Neural Network known to perform well on MNIST (McMahan et al. [2017]). We are considering a cross-silo scenario, so there will be no client failures and we assume every client has a sufficient compute power. A batch size of 8 and learning rate of 0.03 was used in every experiment for consistency.

# 5 Results and discussion

## 5.1 IID MNIST subset experiment

Table 1: Final accuracies over 3 experiments on the MNIST test set, trained on a small IID subset

| Strategy | Range | Median |
|---|---|---|
| Centralised | 89.83 - 89.87 | 89.84 |
| FedAdagrad | 89.30 - 90.24 | 89.54 |
| FedDC (b=10) | 88.91 - 89.42 | 89.33 |
| FedDC (b=5) | 88.81 - 89.14 | 89.04 |
| FedAdam | 88.14 - 88.56 | 88.53 |
| FedYogi | 88.16 - 88.53 | 88.51 |
| FedAvg | 87.84 - 88.10 | 88.02 |

We see in Table 1 that FedDC performs rather well on the IID MNIST subset. Only centralised training has a higher minimum accuracy than FedDC's maximum. It outperforms basic FedAvg by a reasonable amount. FedAdagrad did have a higher median, however.

The results do not fully align with previous work. Previously, FedAdagrad performed the worst in this client scenario however now it performs the best of all federated learning strategies. One theory is that the FedAdagrad implementation used within the paper was either incorrect or used a poor choice of parameters. An additional explanation could be that FedAdagrad just worked better with this particular dataset, it was not previously tested on MNIST. Finally, it is possible that we would see FedDC perform better with more tests, as the interval between the minimum and maximum performance from each strategy do overlap.

However, what does match previous work (and confirms the hypothesis) is that FedDC performs very well in this scenario, almost matching the centralised gold standard. The minimum accuracy achieved by FedDC with any aggregation period is above the maximum achieved by any other strategy excluding FedAdagrad.

We see that the aggregation period does not have much of an impact, this is a common theme between all experiments and is discussed in 5.4.1.

## 5.2 Non-IID MNIST subset experiment

Table 2: Final accuracies over 3 experiments on the MNIST test set, trained on a small non-IID subset

| Strategy | Range | Median |
|---|---|---|
| FedAdagrad | 74.74 - 76.58 | 75.95 |
| FedDC (b=10) | 75.81 - 76.26 | 75.94 |
| Centralised | 75.85 - 75.94 | 75.92 |
| FedDC (b=5) | 75.48 - 76.06 | 75.56 |
| FedAdam | 73.02 - 75.11 | 74.78 |
| FedYogi | 72.98 - 75.08 | 74.70 |
| FedAvg | 72.58 - 73.72 | 73.54 |

Table 2 shows us that FedDC performs excellently in this scenario, even matching the centralised gold standard. FedAdagrad once again performs very well too, also matching centralised training. While FedAdagrad, FedDC and centralised training all perform very similarly, there is a large performance gap between them and the other strategies.

These results match more closely with experiments done previously, however, the other strategies do perform far better than expected (potentially due to better parameter choices/implementation issues as described previously).

The results show that FedDC can learn very effectively when faced with label distribution skew, confirming the hypothesis and validating previous experiments. Further discussion into why FedDC performs well will be expanded upon in 5.4.

### 5.3 Quantity and label skew experiment

Table 3: Final accuracies over 3 experiments on the MNIST test set, trained on a subset of clients with label distribution and quantity skew

| Strategy | Range | Median |
|---|---|---|
| Centralised | 98.26 - 98.59 | 98.48 |
| FedYogi | 98.29 - 98.44 | 98.43 |
| FedAdam | 98.04 - 98.33 | 98.31 |
| FedAvg | 98.18 - 98.35 | 98.28 |
| FedDC (b=5) | 98.12 - 98.27 | 98.23 |
| FedDCW (b=10) | 98.06 - 98.31 | 98.23 |
| FedDCW (b=5) | 98.09 - 98.34 | 98.18 |
| FedDC (b=10) | 98.07 - 98.30 | 98.10 |
| FedAdagrad | 96.00 - 97.33 | 96.78 |

Table 3 shows that under this scenario, most strategies perform very similarly and essentially match centralised training. The only exception is FedAdagrad, which underperforms by quite a large margin.

This result was very surprising. The hypothesis was that the unweighted averaging performed in FedDC, along with a large number of clients with poor-quality data, would result in a significantly lower accuracy than other strategies. My proposed improvement, FedDCW, performed essentially the same as FedDC despite my theory that weighting the updates towards those that had experienced clients with more samples would improve performance.

This surprising performance could be explained by the fact that despite training on mostly poor-quality clients, any set of parameters was very likely to be trained on a client with a massive amount of very good data before aggregation. The negative impact of the poor clients was overestimated, and the positive impact of the good clients was underestimated. Hence, the final result was a very good accuracy.

Although accuracy did increase more quickly as shown in Figure 3, unfortunately, modified weighted averaging (FedDCW) as an aggregation operator for FedDC made no difference to the overall final performance. A potential cause of this is that each set of parameters has the same expected number of samples seen before aggregation, meaning a weighted average has a minimal impact especially as the aggregation period gets larger.

### 5.4 Why does FedDC perform well?

A theory as to why FedDC has shown to be effective is that the act of daisy-chaining emulates training on multiple batches from the global training data, rather than being restricted to batches from a single client. If we have IID clients, then one set of parameters gets exposed to a greater amount of data before aggregation which is very useful if our sample sizes are small. If our clients are non-IID, we get exposure to different types of data to train on before aggregation which is also beneficial.

A key point to note that holds back performance is that each batch from a client will always be from the same distribution. This means we can't quite reach centralised performance when we have a large amount of data heterogeneity since every item within a batch is not independent.

#### 5.4.1 Aggregation period considerations

We see that an aggregation period of 10 performs essentially the same as an aggregation period of 5. Aggregating too early would reduce the amount of data parameters are exposed to before averaging, which would reduce the quality of the model. However, if we leave aggregation too long the parameters might be too different and the resulting aggregated model quality could be poor. It is

possible that the optimal aggregation period depends on factors such as the number of clients or on the data itself. It appears that an aggregation period of 10 was too high considering we essentially reach the same accuracy when the period is 5, perhaps we could've obtained even better results with a slightly lower aggregation period.

### 5.5 Convergence discussion

#### 5.5.1 Convergence curves when training on IID data

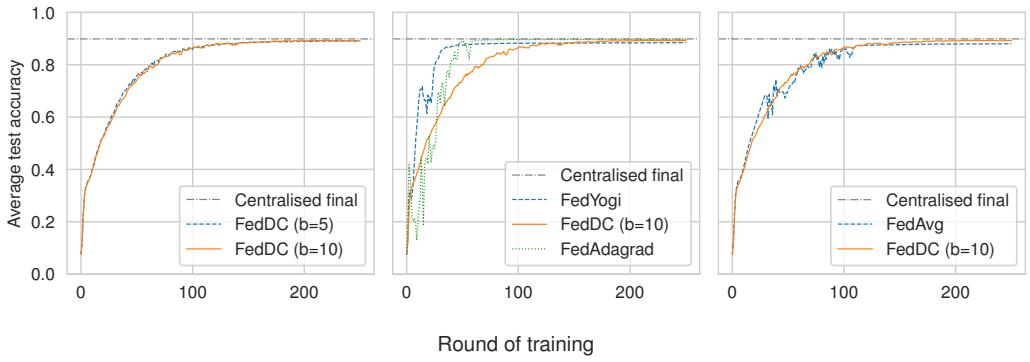


Figure 1: Comparison of the convergence curves of various strategies when trained on an IID subset of MNIST against FedDC

We see in Figure 1 that the training characteristics of FedDC with both tested aggregation periods are essentially the same in this scenario. FedAdam also had essentially the same patterns as FedYogi and was hence omitted.

FedDC's accuracy increases at roughly the same rate as FedAvg, however, it keeps increasing for longer resulting in a higher end result. Convergence is noticeably slower than all other strategies in this scenario. Both FedYogi and FedAdagrad converge roughly 100 rounds earlier than FedDC.

Training is noticeably more stable than any of the other strategies. This could be beneficial if we are limited by the number of rounds we can train for as our model is unlikely to decrease in quality significantly before converging.

#### 5.5.2 Convergence curves when training on non-IID data

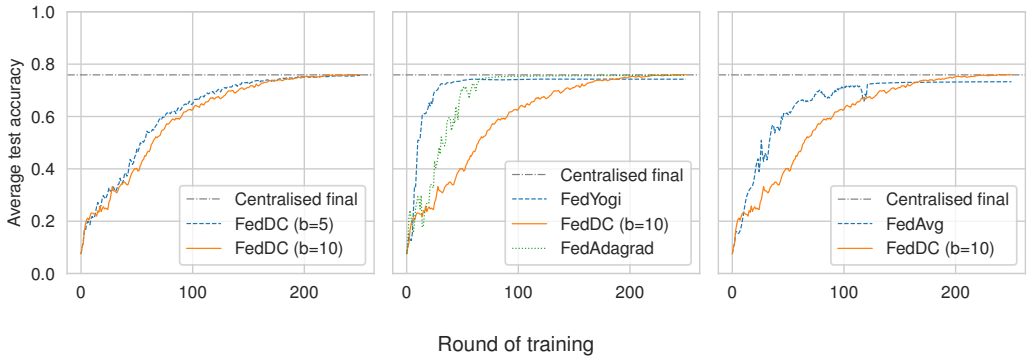


Figure 2: Comparison of the convergence curves of various strategies when trained on a non-IID subset of MNIST against FedDC

On non-IID data, we see very different results from our previous scenario which is quite surprising.

Unfortunately, against every other strategy, including FedAvg, FedDC took significantly more time to converge. A theory is that by training on clients that are so different before aggregation, the magnitudes of our parameters to aggregate are decreased since each round of training pulls them in different directions, this would slow convergence. This theory also explains why a lower aggregation period resulted in a quicker initial increase in accuracy, our updates were trained on fewer clients before aggregation which meant their magnitude was not decreased as much. Finally, we once again see that training via FedDC is incredibly stable even with clients with such different data.

### 5.5.3 Convergence with quantity and label distribution skew

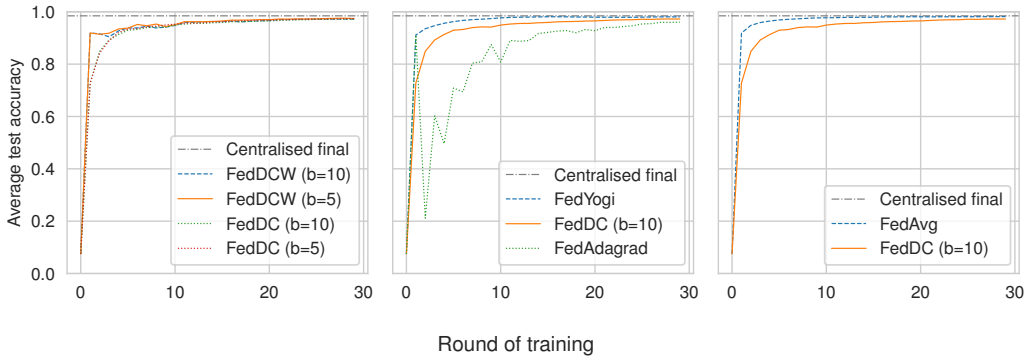


Figure 3: Comparison of the convergence curves of various strategies when trained on clients containing MNIST data with quantity and label distribution skew, against FedDC.

Interestingly, the trends seen in Figure 3 are similar to both previous experiments in some aspects.

There is virtually no difference in the characteristics for each aggregation period again this time, which matches the IID experiment. This is surprising since we have label distribution skew, we would expect to see similar characteristics to Figure 2. The most important training is done on IID clients though, which could be why we see similar aggregation period patterns to Figure 1 instead.

However, the difference in time to converge from the other strategies does match what we saw in the non-IID subset experiment, FedDC converges slower than all other strategies again (except FedAdagrad, which performed extremely poorly overall in this scenario). This implies that non-IID client data slows down FedDC, but does not stop it from learning well. Finally, the performance of FedDCW was disappointing. While the accuracy did initially increase far quicker, converging to a final accuracy took the same time as FedDC. This shows us that the reason for the long convergence time is related to the daisy-chaining phase, not the aggregation phase.

## 6 Conclusion

In conclusion, this report summarises how the FedDC strategy was implemented into the Flower framework. It validates that the strategy is effective on clients with an identical, low amount of either IID or non-IID (with label distribution skew) data. A previously untested scenario with both quantity and label distribution skew is tested which shows that even in this challenging environment, FedDC surprisingly still performs well. The proposed improvement to FedDC's aggregation unfortunately does not produce any significant improvements in the scenarios tested.

The additional analysis on convergence times shows that FedDC takes longer to converge when there is label distribution skew than other strategies, however, the final accuracy remains high. This could limit its use when compute power or communication is limited, but in the proposed use-case of a cross-silo environment with limited amounts of data at each client, this should not be an issue. The issue of privacy was not discussed in this report, however, since a client's model update could be shared with any other client the risk of an attack is higher than with other strategies. The paper describes clipping and adding noise, but this may not be sufficient to protect against all attacks. Further research is needed before the method could be deployed.

## References

Michael Kamp, Jilles Vreeken, and Jonas Fischer. Federated learning from small datasets. In *ICLR*, 2023.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017. URL `https://proceedings.mlr.press/v54/mcmahan17a.html`.

## Appendices

## A Experiment reproducibility

Care has been taken to ensure the experiments performed are reproducible. Within the source code (project/conf) are 4 .yaml files, each corresponding to an experiment or set of experiments. Running project.main with poetry and these config files will carry out the same experiments described within this report and save the results. The graphs and tables can be produced with the data_analysis notebook contained within the plotting subfolder.

mnist_8_samples.yaml runs three experiments on an IID partition of the MNIST train set assigning 8 samples to 25 clients. It runs these with three different client selection seeds, and on six different strategies. It also runs the same experiments on the non-IID partition of the MNIST train set.

mnist_8_samples_centralised.yaml runs the same experiments on both datasets as described previously, but with centralised training instead. Therefore, the client selection seed is not changed.

mnist_large_and_small.yaml again runs three experiments. This time assigning 8 samples to 20 clients and 1200 samples to 5 clients. Three client selection seeds are used again. This time two extra strategies are tested, those being FedDCW with b=5 and b=10.

Finally, mnist_large_and_small_centralised.yaml runs using the same dataset described within the previous experiment, but with centralised training.

All experiments use the same seed for creating the dataset partition dynamically to ensure we are comparing the performance on the same data each time.

## B Change of project goal

This project was initially going to be a Flower Baseline that reimplemented all experiments within the FedDC paper. However, after dedicating a fair amount of work to this I determined I did not possess the compute power to be able to perform these in a reasonable amount of time. Hence, the decision was made to instead perform similar smaller experiments and also perform my own different experiments and analysis so that the scope of the project wasn't too small.