# Why go full, when you have momentum?

## Accelerating layerwise training with momentum

**Simon Istvan Virag<siv24@cam.ac.uk> Asbjorn Lorenzen <at2124@cam.ac.uk>**

## Abstract

Federated learning has emerged as a compelling approach for decentralized training in heterogeneous environments with varying network conditions. Recent work on partial network updates has shown that updating only selected layers —rather than the full model— can improve the layer mismatch issue when client models are aggregated and improve both convergence speed and resource efficiency of the training rounds in federated learning. In this project, we reproduce experiments of the original work under ideal conditions and non-i.i.d. data distributions and further investigate the methods' efficiency when it is impacted by partial participation by activating only a subset of clients per round. Furthermore, we extend the original partial update framework by introducing a layer-wise momentum aggregation mechanism that enhances gradient alignment across layers. With our extended approach we aim to further improve convergence of partial network updates and reduce the amount of communication needed for model training while providing robustness against data heterogeneity and partial participation. Finally, we introduce a novel algorithm for aggregating weights based on global optimizer state and evaluate this in the layer-wise training setting.

## 1 Introduction

Federated learning (FL) is a distributed computing paradigm that facilitates machine learning across a diverse set of data sources/clients without requiring direct data sharing, thereby enhancing privacy, scalability and adaptability of machine learning models in heterogeneous environments. Since FL also involves clients with communication restrictions, optimizing model updates to minimize bandwidth usage and computational overhead is crucial for ensuring efficient training across diverse network conditions. Traditional FL methods, such as FedAvg[11], update the entire model at each communication round which might not be possible for some clients, and often lead to what is known as the "layer mismatch" problem. Layer mismatch occurs when layers in various clients suffer from misalignment between the learned features, ultimately resulting in slower convergence and suboptimal performance. To address this challenge, a recent study by Wang et al. [16] proposed a method for partial network updates -where only a subset of layers are updated at each round— to improve the synchronisation of layer-wise feature extraction and reduce both communication and computational overhead on the client side.

Building on the method of Wang et al., our work introduces three key extensions. First, while the original experiments already included tests with non-i.i.d. data, we extend the analysis by incorporating partial participation, where only a subset of the clients are active in each round, and evaluate the approach on additional model architectures. These modifications provide further insights into the method's robustness under constrained participation and diverse model settings. Second, we propose and evaluate FedPartProx and FedPartMoon, which combine the partial layer training from FedPart with the two well-known algorithms. Thirdly, we incorporate a layer-wise momentum aggregation scheme. By aggregating momentum information separately for each layer, our method

Preprint. Under review.

aims to capture the nuanced gradient dynamics inherent to different parts of the network, leading to more stable updates and accelerated convergence with minimal communication increase. Our method explores the impact of both weighted and unweighted average momentum on convergence speed and model accuracy across different layers. Additionally, we implement a method that compares the momentum of a layer to the momentum of the same layer gathered during full network updates so that we can also control the deviation of the layer from the global optimization trends similarly to FedProx or FedMoon.

# 2 Background

## 2.1 Selective parameter updates

FedDrop [18] adapts the classic dropout technique by generating personalized subnets with heterogeneous dropout rates that are tailored to each client's channel conditions and resource constraints. Only these pruned subnets are sent to devices, which then train these smaller models locally. SpaFL [7] uses structured sparsity by defining trainable thresholds for each filter/neuron to prune connected parameters, creating an automatic sparsification of the models.
In a similary vein, [15] and [19] only share low-rank spaces of parameter matrices. However, these approaches result in performance degradation in non-iid scenarios. LotteryFL [8] makes each client learn a lottery ticked network (a sub-network of the base model) by applying the Lottery Ticked hypothesis [4], which states that within a randomly initialized dense neural network, there exists a subnetwork that - when trained in isolation - can match or exceed the performance of the original full network. However, finding this winning ticket requires significant computational overhead.
Layer mismatch is a common issue in FL that occurs when local models develop specialized feature representations in specific layers, such that the aggregation step causes misalignment between features learned at different layers. Wang et al. [21] first identified this as an issue, particularly in non-iid settings. Various works have attempted to mediate the issue [10] [20], while Collins et al. [3] suggest that lower layers suffer more from this mismatch than higher layers.
FedPart [16] addresses layer mismatch by training only a portion of the neural network in each communication round instead of the entire model. The proposed algorithm updates model layers sequentially from shallow to deep, and then cycles back to shallow layers for multiple training rounds. Each client trains only the designated layers while keeping other parameters frozen, which serves as an anchor that constrains update directions. The approach effectively mitigates the problem where averaged layers struggle to cooperate after federation, allowing the global model to converge more efficiently with better accuracy compared to traditional full network updates.
Of course, only training a single layer at a time potentially requires many more training rounds, which can be problematic in network environments with high latency, which is not taken into account in their experiments. Another crucial issue with FedPart is that the paper assumes full client availability and neglects client dropout, which is unrealistic in a real FL setting. Furthermore, their evaluation under data heterogeneity is limited and shows decreased performance in heterogeneous settings.

## 2.2 Aggregated optimizer states

Recent attention has been given to the approach of aggregating optimizer states along with model parameters to improve convergence. Reddi et al. [13] introduced FedOpt, showing empirical improvements over FedAvg when using adaptive server optimizers on various tasks with heterogeneous data. Since then, numerous other works have incrementally improved on distributed adaptive optimizers and confirmed its effectiveness in diverse FL scenarios [5] [17] [1]. Crucially, work by Cheng et al. [2] provides the first theoretical proof that Local SGD with momentum (Local SGDM) and Local Adam can outperform their minibatch counterparts in certain regimes. For Local SGDM, they establish convergence guarantees in convex settings, while for Local Adam, they prove benefits in weakly convex settings.
While these studies demonstrate the benefits of aggregating optimizer states to improve convergence, they have not yet been extensively explored in the context of partial network updates. The momentum aggregation mechanisms proposed in these works primarily focus on full model updates, leaving a gap in understanding how layer-wise momentum aggregation could specifically benefit partial update frameworks like FedPart.

### 2.3 Adaptive client weighting

Some attention has also been given to weighting the client updates depending on client performance. FedProx [14] presents a simple approach to this by penalizing local models from deviating too far from the global model. SCAFFOLD [6] suggests using control variates to estimate the update direction of clients and corrects local updates to better align with the global objective. Techniques such as Adaptive Federated Averaging [12] adjusts the weight of client updates based on learning the quality of model updates provided by each participant during training using a Hidden Markov Model. Meanwhile, q-FedAvg [9] weights client updates by instead prioritizing updates from clients with a higher loss to improve fairness. However, there exists a research gap in using the optimizer states for weighting client updates. We propose bridging adaptive aggregation methods and optimizer state synchronization by both synchronizing optimizer states between clients and using the differences in optimizer states as proxy for whether clients are updating in the same direction. This should ensure that clients aligned with the global training dynamics contribute more, reducing the impact of divergent updates caused by non-IID data. This promotes faster convergence and a more stable global model by prioritizing consistent and reliable updates.

## 3 Methods

### 3.1 Partial Updates

For partial updates, we implemented partial network updates as described in FedPart [16]. Namely, we created a schedule for layer freezing that initially trains the full network for 2 rounds, and then cycles through the entire network training one layer at a time and freezing the rest. During freezing, clients only receive the updated parameters from the layer that was updated in the previous round, and only need to broadcast their local changes to the unfrozen layer, greatly reducing communication costs. Due to this limitation of communication, FedPart will naturally struggle in cases with low client availability, as clients that dropped out previously will have outdated parameters in these layers. Conveniently, client dropout was also excluded from the experiments [16]. This could be mediated by ensuring that clients each round receive exactly the parameters for any outdated layer, but as this is an improvement on FedPart and we primarily seek to use it as a baseline, we decided to use FedPart exactly as originally formulated.
Our goal with partial network updates is two-fold: First, we seek to compare it to regular full network training with regards to communication and computation costs and the influence of layer freezing on model convergence. Second, we investigate how well FedPart works when combined with other algorithms, namely with FedProx and FedMoon, as well as in scenarios where we aggregate the optimizer states and use these for weighting client updates (as described in the following sections).

### 3.2 Aggregated momentum

For globally aggregating the momentum, we use Local Adam as formulated by [2]. Between rounds, clients share their optimizer states along with the gradients. In the case of Adam, this is the tensor containing the first and second order momentum, which are computed as per usual in Adam. The server then aggregates these states by simply computing the mean values. More formally, for $M$ total workers indexed $m \in [M]$, local steps indexed $k \in [K]$ and communication round $r$, we let $u^m_{r,k}$ denote the 1st order momentum in round $r$ at local step $k$ in worker $m$, and define the 2nd order momentum $v^m_{r,k}$ similarly. The updated states for round $r+1$ before the first step are then computed as

$$u_{r+1,0} = \mathbb{E}_m[u^m_{r,K}]$$
$$v_{r+1,0} = \mathbb{E}_m[v^m_{r,K}]$$

Finally, this new optimizer state is shared with the clients along with the updated model parameters.

Note that when only training a single or few layers, we only need to share the optimizer states and gradients from the layers currently being updated. The sharing of optimizer states imposes a significant added communication overhead, as the Adam optimizer state has twice the size of the related gradients. For FL settings with limited communication bandwidth, this could be a significant hurdle, and this overhead will always be a tradeoff with the improved convergence. The full Local Adam algorithm as described by Cheng et al. [2] can be seen below.

**Algorithm 1** Local Adam

**Require:** initial model $x_0$, learning rate $\eta$, momentum $\beta_1, \beta_2 \in [0, 1)$

Set $x^m_{0,0} = x_0$, $u^m_{0,-1} = 0$, $v_0 = 0$ for each worker $m \in [M]$
**for** $r = 0, \cdots, R-1$ **do**
  **for** each worker $m \in [M]$ in parallel **do**
    **for** $k = 0, \cdots, K-1$ **do**
      $g^m_{r,k} = \nabla F(x^m_{r,k}; \xi^m_{r,k})$, $\widehat{g}^m_{r,k} = \mathbf{clip}(g^m_{r,k}, \rho)$ ▷ Compute clipped stochastic gradient
      $u^m_{r,k} = \beta_1 u^m_{r,k-1} + (1-\beta_1)\widehat{g}^m_{r,k}$ ▷ Update 1st-order momentum
      $v^m_{r,k} = \beta_2 v^m_{r,k-1} + (1-\beta_2)\widehat{g}^m_{r,k} \odot \widehat{g}^m_{r,k}$ ▷ Update 2nd-order momentum
      $x^m_{r,k+1} = x^m_{r,k} - \frac{\eta}{\sqrt{v^m_{r,k}+\lambda^2}} \odot u^m_{r,k}$ ▷ Update model
    **end for**
  **end for**
  $x^m_{r+1,0} = \mathbb{E}_m[x^m_{r,K}]$, $u^m_{r+1,-1} = \mathbb{E}_m[u^m_{r,K-1}]$, $v^m_{r+1,-1} = v_{r+1} := c$ ▷ Communicate and average
**end for**

### 3.3 Weighting updates by similarity between momentums

We propose a novel algorithm that uses the momentums collected from each client to weight the aggregated steps. The intuition behind this idea is that clients whose momentums are similar to the global momentum are updating more in the direction of the global model, so if we weight the updates to the parameters from these clients higher, we might get faster convergence.

In each round, we share the local optimizer states with the server and compute the mean optimizer states $\mathbb{E}_m[u^m_{r,K-1}]$ and $\mathbb{E}_m[u^m_{r,K-1}]$ as described in the previous section. To compute the cosine similarities between client optimizer states and the global optimizer state, we concat the first and second momentum as $z_r = (\mathbb{E}_m[u^m_{r,K-1}] \| \mathbb{E}_m[v^m_{r,K-1}])$ and $z^m_{r,K-1} = (u^m_{r,K-1} \| v^m_{r,K-1})$. After that, the cosine similarities between the concatenated individual client optimizer states and the concatenated mean state are computed as

$$\text{similarity}(z^m_{r,K-1}, z_{r+1,0}) = \frac{z^m_{r,K-1} \cdot z_{r+1,0}}{\|z^m_{r,K-1}\| \cdot \|z_{r+1,0}\|}$$

.

Finally, the updated parameters $\theta$ and the updated first and second momentums $u_{r+1,0}, v_{r+1,0}$ are calculated by summing them weighted by cosine similarity as

$$\theta_{r+1} = \frac{\sum_{m=0}^{M} \text{similarity}(z^m_{r,k}, z_{r+1,0}) \cdot \theta^m_r}{\sum_{m=0}^{M} \text{similarity}(z^m_{r,k}, z_{r+1,0})}$$

$$u_{r+1,0} = \frac{\sum_{m=0}^{M} \text{similarity}(z^m_{r,k}, z_{r+1,0}) \cdot u^m_{r+1,0}}{\sum_{m=0}^{M} \text{similarity}(z^m_{r,k}, z_{r+1,0})}$$

$$v_{r+1,0} = \frac{\sum_{m=0}^{M} \text{similarity}(z^m_{r,k}, z_{r+1,0}) \cdot v^m_{r+1,0}}{\sum_{m=0}^{M} \text{similarity}(z^m_{r,k}, z_{r+1,0})}$$

When doing partial updates, the cosine similarity is only computed on the un-frozen layers of the optimizer state, as we otherwise will always have almost-identical optimizer states only differing in the non-frozen layer. The full algorithm can be seen below.

## 4 Experimental setup and methodology

In this section, we outline the experimental design used to evaluate our approach. Due to the lack of computational resources, each experimet uses the CIFAR-10 dataset to train a simple Convolutional Neural Network shown in Listing 1 on 6 clients. This network is designed to emphasize layer mismatch problem by using many fully connected layers where the parameters are more likely to mis-

**Algorithm 2** Similarity-Weighted Local Adam

**Require:** initial model $x_0$, optimizer states $u_0 = 0$, $v_0 = 0$

**for** $r = 0, \cdots, R-1$ **do**
  **for** each worker $m \in [M]$ in parallel **do**
    Perform $K$ local Adam updates to obtain $x^m_{r,K}$, $u^m_{r,K-1}$, $v^m_{r,K-1}$
  **end for**
  $\bar{u}_r = \mathbb{E}_m[u^m_{r,K-1}]$, $\bar{v}_r = \mathbb{E}_m[v^m_{r,K-1}]$ ▷ Compute mean optimizer states
  $z_r = (\bar{u}_r \| \bar{v}_r)$ ▷ Concatenate mean optimizer states
  **for** each worker $m \in [M]$ **do**
    $z^m_r = (u^m_{r,K-1} \| v^m_{r,K-1})$ ▷ Concatenate worker optimizer states
    $s^m_r = \frac{z^m_r \cdot z_r}{\|z^m_r\| \cdot \|z_r\|}$ ▷ Compute cosine similarity
  **end for**
  $S_r = \sum_{m=1}^{M} s^m_r$ ▷ Sum of similarities
  $x_{r+1,0} = \frac{\sum_{m=1}^{M} s^m_r \cdot x^m_{r,K}}{S_r}$ ▷ Update weighted model parameters
  $u_{r+1,-1} = \frac{\sum_{m=1}^{M} s^m_r \cdot u^m_{r,K-1}}{S_r}$ ▷ Update weighted 1st-order momentum
  $v_{r+1,-1} = \frac{\sum_{m=1}^{M} s^m_r \cdot v^m_{r,K-1}}{S_r}$ ▷ Update weighted 2nd-order momentum
  Distribute $x_{r+1,0}$, $u_{r+1,-1}$, $v_{r+1,-1}$ to all workers
**end for**

align to demonstrate the benefit of layer-wise training, as well as to be computationally lightweight so that multiple clients can be used with it on a single M2 Macbook machine. The CIFAR-10 dataset was chosen as it provides enough complexity to demonstrate layer mismatch issues while remaining computationally tractable for multi-client simulation on limited hardware. Also, its moderate size and balanced class distribution make it ideal for controlled non-IID experiments (using Dirichlet partitioning).

Listing 1: Implementation of the testing CNN

```
class Net(nn.Module):
 def __init__(self) -> None:
     super(Net, self).__init__()
     self.conv1 = nn.Conv2d(3, 6, 5)
     self.pool = nn.MaxPool2d(2, 2)
     self.conv2 = nn.Conv2d(6, 16, 5)
     self.fc1 = nn.Linear(16 * 5 * 5, 180)
     self.fc2 = nn.Linear(180, 160)
     self.fc3 = nn.Linear(160, 140)
     self.fc4 = nn.Linear(140, 120)
     self.fc5 = nn.Linear(120, 84)
     self.fc6 = nn.Linear(84, 10)
```

Each training round completes 8 epochs in all clients, and the clients each have their own optimizer states as default, which is reset between rounds. We use the Adam optimizer implemented in PyTorch with the default hyperparameters, and we used a batch size of 32. For FedMoon, we used a temperature parameter of $0.5$, and for FedProx, we used a proximal $\mu$ of $0.1$. For the strategies using momentum aggregation, the clients share the optimizer state dict for the trained layers along with the new trained parameters, and receive the new globally aggregated optimizer state at the beginning of the next round. The momentum states are globally aggregated using weighted averaging based on either the amount of viewed data (in the 'unweighted' case) or based on cosine similarity between optimizer states (the 'weighted' case). This algorithm is described in greater detail in Section 3.3.

We initialize a cycle of partial network updates by doing two FNU rounds as warm-up. This ensures that the initial global model achieves a stable state before layer-wise updates are applied. Furthermore, each layer undergoes two consecutive training rounds as per the original paper. This brings the total number of rounds to 18 (2 initial plus $2 * 8$ for each of the $8$ layers). For consistency, we therefore also ran FNU experiments on a total of 18 rounds. We implemented the partial network updates by sending the desired trained layers to the client during each round, allowing the client to

freeze all layers but the trained one. We only communicate the trained layer between the server and the client each round, exactly as detailed in the FedPart paper.

To analyze the impact of both layer-wise model updates and layer-wise momentum, we use FedAvg, FedProx, and FedMoon as baseline strategies. FedAvg was included because it represents the standard approach in FL, and comparing our results to it makes it easier to understand the relative performance of all other approaches. We selected FedProx because of its ability to address client drift, which is particularly useful in the client dropout and heterogeneous data scenarios. Finally, FedMoon was included due to its contrastive learning concepts that can improve feature alignment across clients, which addresses the layer mismatch problem that is the target of our methods.

Building on these, we implement FedPartAvg, FedPartProx, and FedPartMoon to facilitate simple layer-wise network updates. Additionally, we introduce momentum-based equivalents—FedPartAvgMomentum (FPAM), FedPartProxMomentum (FPPM), and FedPartMoonMomentum (FPMM) which incorporate layer-wise momentum aggregation to assess its effects on training stability, convergence speed, and model accuracy. This comparative analysis allows us to evaluate the effectiveness of layer-wise updates across different federated learning optimization strategies.

To comprehensively evaluate our proposed methods, we design three distinct testing scenarios:

**Best Performance Conditions:** In our first experiment, we assess the performance of the layer-wise update method under ideal conditions where data is independently and identically distributed (IID) across all clients and all clients are available in each round. This experiment helps to understand the best performance that the given strategy can present with the model and can serve as a baseline to which we can compare our following experiments.

**Partial Participation:** In the second experiment, we introduce partial client participation, where only a subset of the clients are active in each training round. This scenario evaluates the impact of client dropout on layer-wise training dynamics and overall model convergence. By varying the active clients per round, we assess how the model adapts to inconsistent participation and whether layer-wise updates can maintain stability and learning efficiency despite inconsistent client engagement. In order to simulate partial participation we have implemented our custom `ClientManager` where it can be defined what percentage of clients should be sampled randomly from the client-set for a given round. For this report we used 50% drop out value of to simulate low client participation.

**Non-IID Data:** The final experiment investigates the impact of non-IID data distributions on training performance. By assigning clients data that is not uniformly distributed, we examine how layer-wise updates compare to traditional full-network updates under heterogeneous data conditions. This experiment aims to simulate clients with unique data distributions to reflect real-world federated learning scenarios where data is inherently non-uniform across clients. To achieve this, we utilize the Flower framework's `DirichletPartitioner` [22], which allows controlled partitioning of the dataset based on a Dirichlet distribution, ensuring varying levels of data heterogeneity among clients. For this report we used the Drichlet value of 0.1 to simulate extreme data heterogeneity. This means that it is common for a single client to posses 80-90% of the data with a particular label, making convergence extremely difficult. For a visualisation of this, see Appendix Figure 5.

To facilitate comparison, each experiment presents the baseline performance of the original strategies and their FedPart variants, followed by an analysis of the momentum-based implementations. We focus on comparing FedProx and its variants in the main experiments; comparison plots for the other strategies are provided in the Appendix.

In each of the experiments, we measure the communication cost (megabytes transmitted in a round),the computational cost (FLOPS), and the global model accuracy. The FLOPS are the calculated costs of processing a single image through the model including both forward and backward pass, so they will scale with the size of the dataset the clients are using. The FLOPS used were estimated using the `ptflops` library, and the difference between FLOPS is due to the partial update setting only performing backward-pass on the trained layer while the FNU approach performs a backward pass on all layers. Communication costs were gathered by measuring the size of the data transmitted between Flower clients and the server.
These metrics allow us to evaluate the efficiency of different update strategies, their impact on convergence speed, and the overall effectiveness of layer-wise updates in federated learning environments. All experiments were repeated three times, and we report the min, max and median average

in each. In our plots, for the sake of clarity and visibility, we only include the median value. To ensure reproducibility, we have included details about library versioning in the associated GitHub repository.

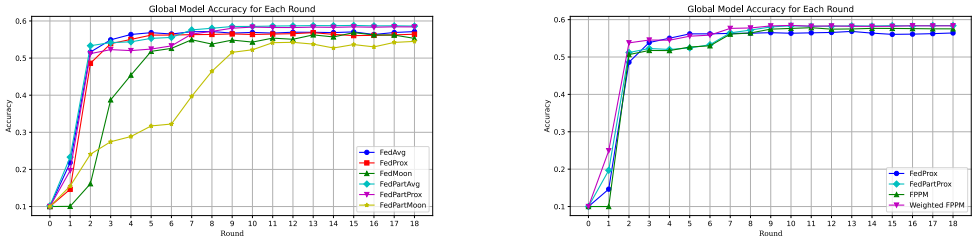
# 5 Results and Discussion

## 5.1 Best Performance Conditions

An overview of our results can be seen in Table 1. Notably, we see that the communication cost is greatly decreased when using partial layer training, but is increased again when sharing the optimizer state, which for first and second momentum is 2x the size of the parameters. In the case of FedMoon and FedProx their layer-wise implementations of FedPartMoon and FedPartProx present a better final accuracy while FedAvg and FedPartAvg remain within the vicinity of each other. Meanwhile, the combination of FedPart and aggregated momentum greatly increases accuracy in the ideal setting.

| Strategy | Comm. Cost | Comp. Cost | Min Acc. | Median Acc. | Max Acc. |
|---|---|---|---|---|---|
| FedAvg | 127.8 | 91,552 | 56.42 | 57.14 | 57.89 |
| FedProx | 127.8 | 91,552 | 55.76 | 56.43 | 57.15 |
| FedMoon | 127.8 | 91,552 | 54.68 | 55.31 | 55.97 |
| FedPartAvg | 31.9 | 41,896 | 57.95 | 58.63 | 59.21 |
| FedPartProx | 31.9 | 41,896 | 57.78 | 58.4 | 59.12 |
| FedPartMoon | 31.9 | 41,896 | 53.89 | 54.5 | 55.27 |
| FPAM (unweighted) | 95.7 | 41,896 | 56.19 | 56.83 | 57.52 |
| FPPM (unweighted) | 95.7 | 41,896 | 56.94 | 57.53 | 58.23 |
| FPMM (unweighted) | 95.7 | 41,896 | 54.71 | 55.4 | 56.12 |
| FPAM (weighted) | 95.7 | 41,896 | 58.75 | 59.24 | 59.36 |
| FPPM (weighted) | 95.7 | 41,896 | 58.68 | 59.44 | 59.56 |
| FPMM (weighted) | 95.7 | 41,896 | 54.63 | 55.24 | 55.89 |

Table 1: Results of strategies under best performance conditions




(a) Accuracy of Global Model per round for baseline and FedPart models

(b) Accuracy of FPPM and Weighted FPPM compared to FedProx and FedPartProx

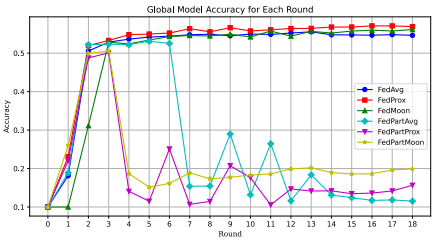
Figure 1: Results of baseline and layer-wise strategies the ideal setting

The results for the ideal setting are found in Figure 1a. We see that FedPartAvg and FedPartProx reach the highest final accuracy. This is explained by the fact that using partial training to avoid layer-mismatch helps with reaching a more stable final result. FedPartMoon and FedMoon have the lowest final accuracy, which is likely due to its contrasting learning not being beneficial in the homogeneous data setting. In this ideal IID setting, the additional optimization goal seems to interfere with the more straightforward classification objective, which causes the lower final accuracy. In Figure 1b, we compare FedProx to the three variations using partial training: the simple FedPartProx, FPPM (aggregating momentum) and weighted FPPM (aggregating momentum and weights based on cosine similarity between optimizer states). We see that all three extensions to FedProx perform better than the baseline, with FedPartProx and Weighted FPPM performing slightly better than FPPM. However, the performance is so close that it could be due to only performing 3 experiments of each (as suggested by the min and max measured accuracies from each method being very close

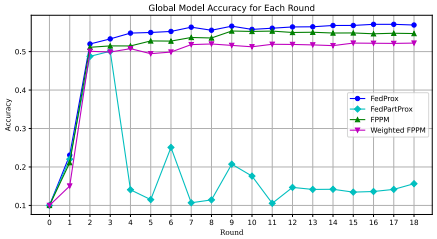
together). Overall, we can conclude that partial updates have a definite positive effect, even in the ideal setting. This is remarkable because partial updates require significantly less communication and computation.

### 5.2 Partial Participation Simulation

In scenarios with partial participation (Figure 2a), strategies that use partial network updates experience a notable performance decline, primarily due to inconsistent layer updates across rounds. With only a subset of clients participating in each round, not all layers are updated uniformly—a layer actively trained in one round may not be updated in the next if the newly selected clients did not previously update that layer, causing the so-called ”anchor” layers to become outdated and diminishing the learning capacity of subsequent layers. This inconsistency worsens a layer mismatch issue, where varying update frequencies lead to misaligned gradients across layers, disrupting synchronization and hampering the global model's convergence. Seeing that this inconsistency leads to such poor results, it would be ideal to suggest an improvement to FedPart that shares the parameters of all outdated layers in the client during each round, as this would avoid the issue. However, as FedPart specifically mentions not doing this, we have excluded it from our experiments. As an extension of these experiments, it would be interesting to see how the sharing of outdated parameters in each round could mitigate this issue, but due to the long time-requirements needed for running experiments, we have not included that in this report.
In Figure 2b, we see that momentum aggregation has a hugely positive effect on the model accuracy in the dropout setting. By sharing the optimizer state, we are effectively transferring knowledge about the direction and magnitude of previous updates, which helps non-participating clients catch up even when they missed rounds. This helps ensure all clients follow a similar optimization path, reducing the divergence between participating and non-participating clients. These results suggest that the sharing of optimizer state is extremely beneficial in this setting where clients, due to dropout, may end up with outdated parameters.




(a) Accuracy of Global Model per round for baseline and FedPart models in partial participation setting




(b) Accuracy of FPPM and Weighted FPPM compared to FedProx and FedPartProx in partial participation setting

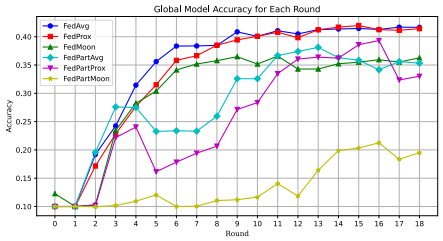
Figure 2: Results of baseline and layer-wise strategies under partial participation

### 5.3 Non-IID Data Scenario

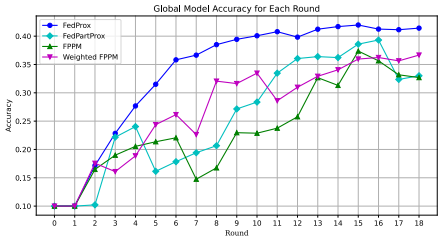
Recall that we used extremely heterogeneous data distribution using the Dirichlet Partitioner with $\alpha = 0.1$, and that an example of the distribution of the data to 6 clients can be found in Figure 5 in the appendix. In Table 3, we see that all models reach a significantly lower convergence of at most $42\%$ accuracy. Surprisingly, the best performance comes from the simple FedAvg and FedProx algorithms. In this setting, it is clearly beneficial to train the model for many more rounds, as we need to slowly aggregate a lot of information about each client's unique data label distribution in order to get a useful global model. This can be confirmed by noticing that the full network update approaches only reach convergence after around 13 full rounds, while the ideal setting reached it after 3 rounds. With extreme data heterogeneity, the partial update approaches may lead to the frozen layers containing representations that are inappropriate for certain clients' distributions, forcing subsequent layers to work with suboptimal feature representations. Meanwhile, the partial updates restrict the model's ability to adapt to these extreme differences between clients, and the frozen layers create a bottleneck preventing the model from capturing the full range of data variations. Recall that the partial update approach only does a single complete sweep through the frozen layers, while the FNU

approach repeats one of these each round. We would likely get much stronger results from completing many more cycles of layer freezing in this setting, and it would be interesting to see how such a final result could compare to the FNU approach. Due to our limited computational resources and the very long training time required to do a large amount of cycles, such an experiment has not been included in this work.

Examining Figure 3 (b), we find that the Weighted FPPM approach generally performs better than the other two counter-parts. This might be due to the aggregation based on optimizer state similarity helping the global model better learn the update directions shared between many clients, which intuitively should prioritize learning feature representations that are generally useful across clients. Finally, we see that the implementation of FedPartMoon in this case performs significantly worse than all other approaches. We believe this is due to a bug in the code for the experiment when combining partial training and FedMoon, but we unfortunately have not had time to re-run the experiment to confirm this. Future work should revisit this and evaluate it properly.




(a) Accuracy of Global Model per round for baseline and FedPart models under extreme data heterogeneity




(b) Accuracy of FPPM and Weighted FPPM compared to FedProx and FedPartProx under extreme data heterogeneity

Figure 3: Results of baseline and layer-wise strategies under data heterogeneity

# 6 Conclusion

Our investigation of partial network updates in federated learning demonstrates promising results, particularly in reducing communication and computational costs while maintaining or improving model performance in ideal conditions.

The introduction of layer-wise momentum aggregation with similarity-based weighting further enhanced convergence, especially when mitigating layer mismatch issues. However, our experiments revealed important limitations in realistic scenarios with partial client participation and extreme data heterogeneity, where traditional full network approaches generally outperformed partial update methods. Future work should expand these investigations to larger, more complex models where layer mismatch issues may be more pronounced, and examine the performance of multiple cycles of layer freezing in heterogeneous settings.

Further research could investigate adaptive parameter transmitting methods, such as only transmitting parameters when their delta exceeds some adaptive threshold as well as adaptive layer selection strategies that dynamically adjust based on client participation patterns and data distributions Our novel approach for using optimizer-similarity-based aggregation would also benefit from a thorough ablation study and further investigations to determine in which situations it is more or less beneficial.

# References


[1] Xiangyi Chen, Xiaoyun Li, and Ping Li. "Toward Communication Efficient Adaptive Gradient Method". In: *CoRR* abs/2109.05109 (2021). arXiv: 2109.05109. URL: https://arxiv.org/abs/2109.05109.

[2] Ziheng Cheng and Margalit Glasgow. *Convergence of Distributed Adaptive Optimization with Local Updates*. 2025. arXiv: 2409.13155 [cs.LG]. URL: https://arxiv.org/abs/2409.13155.

[3] Liam Collins et al. "Exploiting Shared Representations for Personalized Federated Learning". In: *CoRR* abs/2102.07078 (2021). arXiv: 2102.07078. URL: https://arxiv.org/abs/2102.07078.

[4] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Training Pruned Neural Networks". In: *CoRR* abs/1803.03635 (2018). arXiv: 1803.03635. URL: http://arxiv.org/abs/1803.03635.

[5] Sai Praneeth Karimireddy et al. "Mime: Mimicking Centralized Stochastic Algorithms in Federated Learning". In: *CoRR* abs/2008.03606 (2020). arXiv: 2008.03606. URL: https://arxiv.org/abs/2008.03606.

[6] Sai Praneeth Karimireddy et al. "SCAFFOLD: Stochastic Controlled Averaging for On-Device Federated Learning". In: *CoRR* abs/1910.06378 (2019). arXiv: 1910.06378. URL: http://arxiv.org/abs/1910.06378.

[7] Minsu Kim et al. *SpaFL: Communication-Efficient Federated Learning with Sparse Models and Low Computational Overhead*. 2024. URL: https://openreview.net/forum?id=DZyhUXpEee.

[8] Ang Li et al. "LotteryFL: Personalized and Communication-Efficient Federated Learning with Lottery Ticket Hypothesis on Non-IID Datasets". In: *CoRR* abs/2008.03371 (2020). arXiv: 2008.03371. URL: https://arxiv.org/abs/2008.03371.

[9] Tian Li, Maziar Sanjabi, and Virginia Smith. "Fair Resource Allocation in Federated Learning". In: *CoRR* abs/1905.10497 (2019). arXiv: 1905.10497. URL: http://arxiv.org/abs/1905.10497.

[10] Mi Luo et al. "No Fear of Heterogeneity: Classifier Calibration for Federated Learning with Non-IID Data". In: *CoRR* abs/2106.05001 (2021). arXiv: 2106.05001. URL: https://arxiv.org/abs/2106.05001.

[11] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[12] Luis Muñoz-González, Kenneth T. Co, and Emil C. Lupu. *Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging*. 2019. arXiv: 1909.05125 [stat.ML]. URL: https://arxiv.org/abs/1909.05125.

[13] Sashank J. Reddi et al. "Adaptive Federated Optimization". In: *CoRR* abs/2003.00295 (2020). arXiv: 2003.00295. URL: https://arxiv.org/abs/2003.00295.

[14] Anit Kumar Sahu et al. "On the Convergence of Federated Optimization in Heterogeneous Networks". In: *CoRR* abs/1812.06127 (2018). arXiv: 1812.06127. URL: http://arxiv.org/abs/1812.06127.

[15] Haolin Wang et al. "Svdfed: Enabling communication-efficient federated learning via singular-value-decomposition". In: *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE. 2023, pp. 1–10.

[16] Haolin Wang et al. *Why Go Full? Elevating Federated Learning Through Partial Network Updates*. 2024. arXiv: 2410.11559 [cs.LG]. URL: https://arxiv.org/abs/2410.11559.

[17] Jianyu Wang et al. "Local Adaptivity in Federated Learning: Convergence and Consistency". In: *CoRR* abs/2106.02305 (2021). arXiv: 2106.02305. URL: https://arxiv.org/abs/2106.02305.

[18] Dingzhu Wen, Ki Jun Jeon, and Kaibin Huang. "Federated Dropout - A Simple Approach for Enabling Federated Learning on Resource Constrained Devices". In: *CoRR* abs/2109.15258 (2021). arXiv: 2109.15258. URL: https://arxiv.org/abs/2109.15258.

[19] Xinghao Wu et al. *Decoupling General and Personalized Knowledge in Federated Learning via Additive and Low-Rank Decomposition*. 2024. arXiv: 2406.19931 [cs.LG]. URL: https://arxiv.org/abs/2406.19931.

[20] Rui Ye et al. *FedDisco: Federated Learning with Discrepancy-Aware Collaboration*. 2023. arXiv: 2305.19229 [cs.LG]. URL: https://arxiv.org/abs/2305.19229.

[21] Felix X. Yu et al. *Federated Learning with Only Positive Labels*. 2020. arXiv: 2004.10342 [cs.LG]. URL: https://arxiv.org/abs/2004.10342.

[22] Mikhail Yurochkin et al. *Bayesian Nonparametric Federated Learning of Neural Networks*. 2019. arXiv: 1905.12022 [stat.ML]. URL: https://arxiv.org/abs/1905.12022.

# Appendices

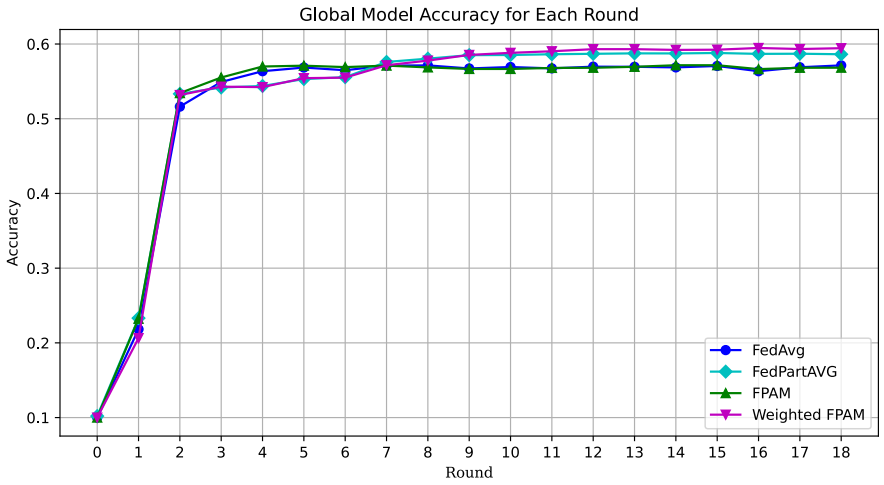
## A Reproducibility

The reproducibility of the project is ensured by the structured organization and standardized setup of the repository. Each strategy has its own dedicated folder, and every experiment is contained within its own Jupyter notebook within that folder, allowing for straightforward execution of individual experiments. To ensure consistency across runs, particularly in data-heterogeneous scenarios, a fixed seed has been set, which guarantees that random processes yield repeatable results. Additionally, we've included a `requirements.txt` file to specify all necessary dependencies, enabling anyone to replicate the exact software environment used in our experiments.

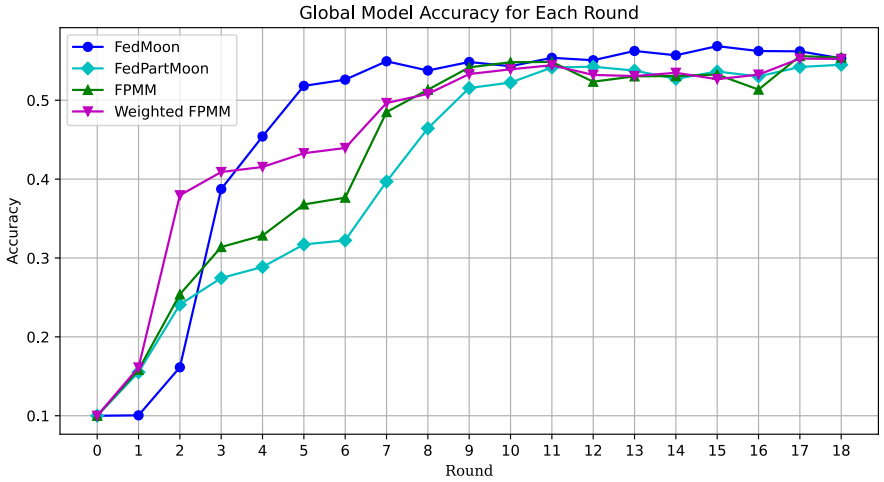
The implementation for all experiments can be found on `https://github.com/ViragSimon/L361_Project`.

# B Additional results using momentum

## B.1 Ideal Scenario




(a) Per round accuracy of FPAM and Weighted FPAM compared to FedAvg and FedPartAvg in ideal setting




(b) Per round accuracy of FPMM and Weighted FPMM compared to FedMoon and FedPartMoon in ideal setting

Figure 4: Comparative results of FedAvg and FedMoon based strategies in an ideal setting

## B.2 Data heterogeneity

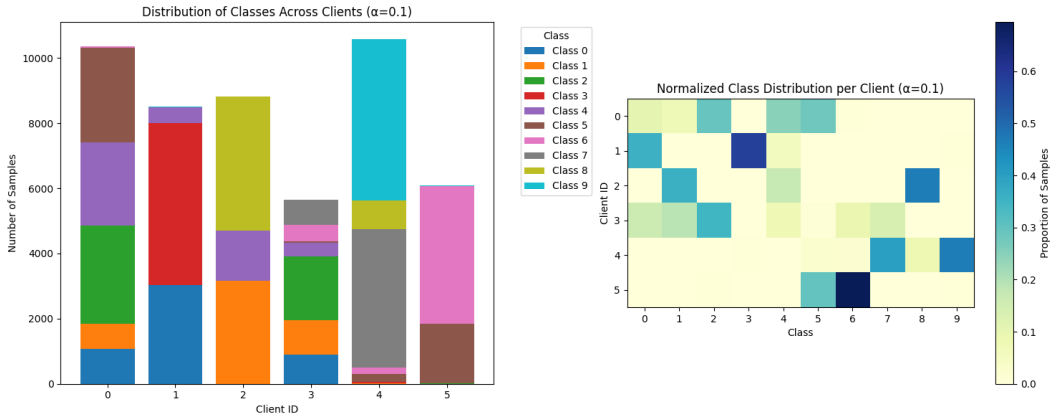
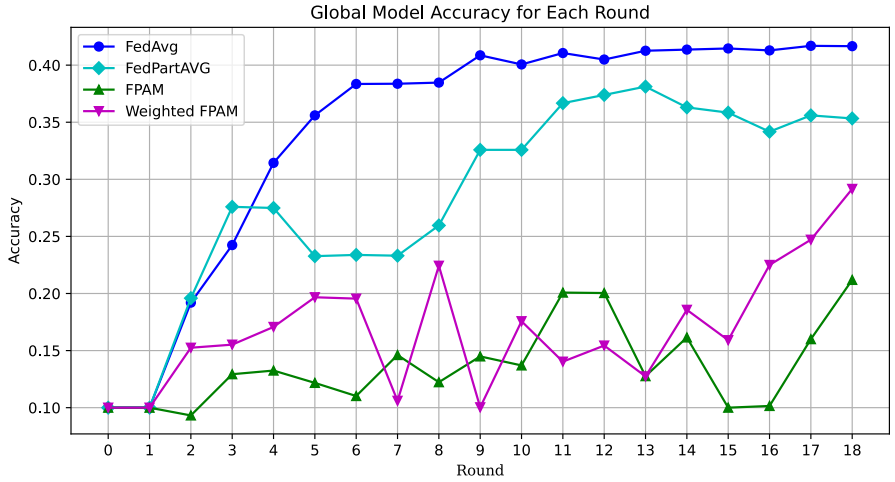
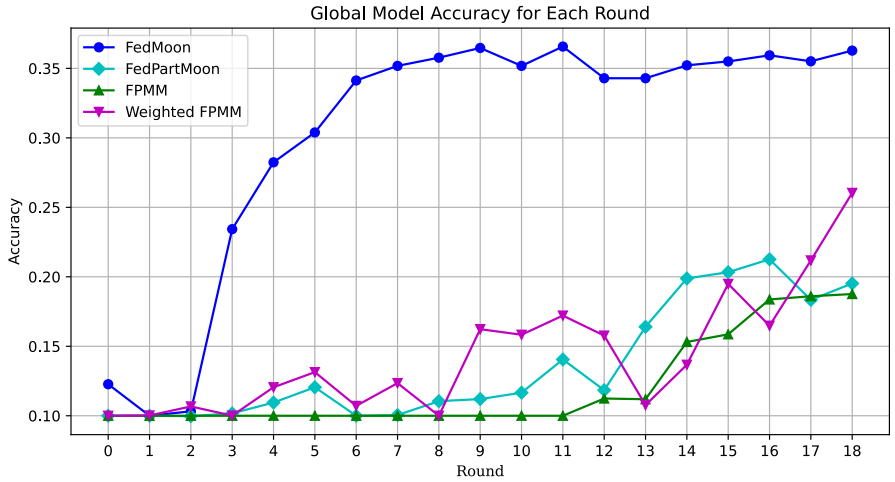



Figure 5: Distribution of data to clients using the Dirichlet partitioner with $\alpha = 0.1$



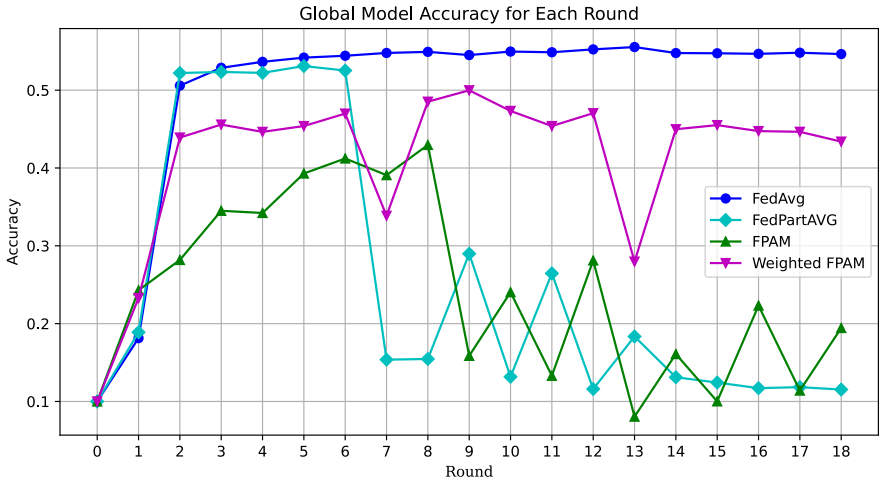

(a) Per round accuracy of FPAM and Weighted FPAM compared to FedAvg and FedPartAvg in a data heterogenous setting



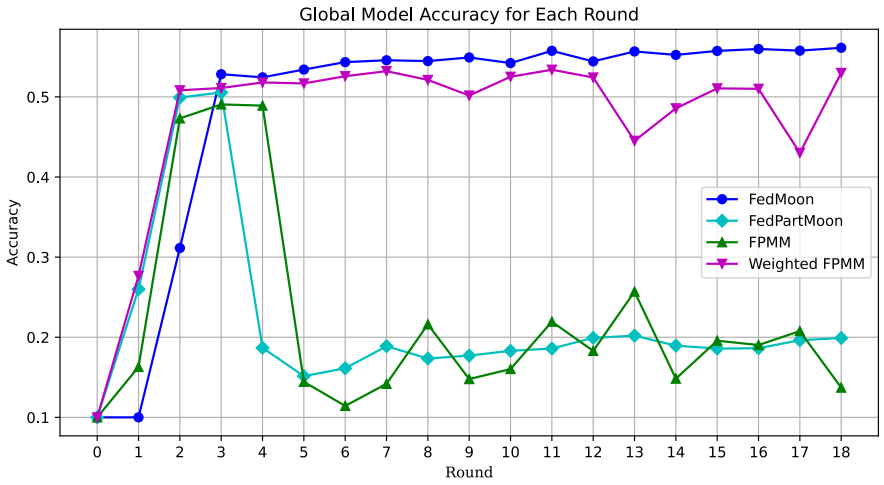

(b) Per round accuracy of FPMM and Weighted FPMM compared to FedMoon and FedPartMoon in data heterogenous setting

Figure 6: Comparative results of FedAvg and FedMoon based strategies in a data heterogenous setting ($\alpha = 0.1$)

### B.3 Partial Participation




(a) Per round accuracy of FPAM and Weighted FPAM compared to FedAvg and FedPartAvg in a partial paticipation setting




(b) Per round accuracy of FPMM and Weighted FPMM compared to FedMoon and FedPartMoon in a partial participation setting

Figure 7: Comparative results of FedAvg and FedMoon based strategies in a 50% partial participation setting