ECE5545 Term Paper, Spring 2024

Ananya Das (ad2258)

Abstract

Federated learning has emerged as a transformative paradigm in machine learning, enabling collaborative model training across decentralized devices while preserving data privacy. In this paper, the basics of federated learning are recapped and various optimization strategies to make federated learning more efficient are explored. Important baseline algorithms such as the Conventional Parameter Server approach and Federated Averaging (FedAvg) are discussed, and then selected optimization strategies proposed in the literature are presented and their implications for convergence speed, communication efficiency, and model performance are analyzed. Specifically, two approaches are discussed in detail: Edge Parameter Server (EdgePS) which dynamically determines the best time points to perform model aggregation, and Resource-Based Aggregation Frequency (RAF), which seeks to optimize federated learning by dynamically adjusting aggregation frequencies based on device resources. The paper also touches upon how some of the techniques presented can be potentially combined together to improve performance even further.

1. Overview of Federated Learning

1.1 Federated Learning

Federated learning is a machine learning approach where a model is trained across multiple decentralized devices or servers holding local data samples, without collecting or exchanging the data centrally. Instead of sending raw data to a central server, the model is sent to the data sources, where it trains on local data, and only the model updates or the trained model parameter values (not the raw data) are sent back to the central server.

Federated learning operates through a network of workers, which are typically edge devices or nodes that possess subsets of the overall dataset and maintain a local copy of the model. These workers conduct local training on their respective data while also interfacing with a parameter server. The parameter server functions as the repository for the parameters of the global model, acting as the central coordinator in the usual setup. It collects updates from all the workers, aggregates these updates, often by averaging gradients, and subsequently updates the global model.

"Local training" refers to the process where each worker trains its model on its local dataset for one or a specified number of epochs. Conversely, "aggregation" denotes the phase of the training process where the updates or parameters from individual workers are combined to refine the global model.

There are various network setups in which federated learning can be performed. In a centralized setup, all workers send their computed updates to a single central server, which aggregates these updates to update the global model and redistributes it back to the workers, posing a bottleneck and single point of failure. Decentralized systems eliminate the central server; instead, workers communicate directly with each other, sharing and aggregating updates to achieve a consensus on the global model, which reduces communication bottleneck but can be slow to converge. A hierarchical setup organizes workers and servers into multiple layers, where lower-tier nodes send their updates to higher-tier aggregation nodes, culminating in a root node that might act similarly to a central server.

1.2 Edge Computing Environments and Heterogeneity

In the context of edge computing, workers differ from each other in various aspects including differences in processing capacity, communication bandwidth, as well as in quantity and distribution of training data. A device becomes a straggler if it has low processing capacity and communication bandwidth while having a high quantity of training data.

Also, for federated learning, the distribution of data among the devices or nodes is often or even usually not identically and independently distributed (non-IID. This non-IID nature arises due to diverse sources of data generation and local data collection processes, leading to variations in data distributions across worker nodes. As a result, federated learning algorithms must contend with the challenge of reconciling these disparate data distributions while collaboratively training a global model, requiring specialized techniques to effectively leverage the decentralized data resources available on edge devices. There are many different ways that the data can be non-IID, and this has implications for the requirements of a federated learning algorithm such that it can still train effectively [1].

1.3 Advantages and Challenges

This decentralized approach allows for training on data distributed across various devices while preserving privacy by keeping sensitive information locally stored.

However, in scenarios characterized by limited network bandwidth, such as edge environments, the time taken to transmit model parameters to the parameter server is often orders of magnitude longer than the duration of one epoch of training. This leads to suboptimal resource utilization, as workers spend considerable time idling while waiting for parameter exchanges. Especially recently, with more edge devices now featuring hardware accelerators like GPUs or specialized processors such as neural processing units (NPUs) found in mobile devices, the training process has become significantly faster, further widening the gap between the time taken for an epoch of local training vs the time taken to send all model parameters from edge to server.

Consequently, the inefficiencies in the conventional parameter server algorithm are magnified, as faster workers are forced to idle while slower ones catch up, hindering overall efficiency.

2. Conventional Methods of Federated Learning

2.1 Conventional Parameter Server Approach

The conventional parameter server approach in federated learning typically involves all devices aggregating their model updates after every epoch of local training. In other words, this approach maintains a consistent alternation between one epoch of local training and one aggregation round for all workers in the system.

However, this method encounters challenges such as being hindered by stragglers, where slower devices can delay the overall training process. Additionally, performing aggregations after every epoch can be computationally expensive and may not always lead to significant improvements in the global model. These drawbacks can lead to inefficiencies in resource utilization and slower convergence speed.

2.2 Federated Averaging

Federated Averaging (FedAvg) is a cornerstone algorithm in the domain of federated learning introduced by Google researchers [2]. The algorithm operates through a series of iterative rounds, wherein a subset of devices is sampled for local training, followed by the aggregation of their model updates at a central server. Notably, FedAvg performs model aggregation by simply averaging the parameters received from participating devices. Importantly, FedAvg works well for the non-IID data distributions practically encountered in real-world federated learning scenarios.

3. Optimizations for Federated Learning

3.1 Theoretical Discussion

3.1.1 Objectives of Optimization

To optimize the process of federated learning, it would require to improve one or both of either the time required to perform the training, or the cost required to perform the training, and in this case the major cost in federated learning on edge devices is the communication bandwidth cost which is incurred in the exchange of parameter values between workers and parameter server(s) during the aggregation phase.

3.1.2 Potential Areas to Address

Model aggregation to combine the local models of each worker to update the single global model (usually at the parameter server) is a time and resource-intensive operation, making this an important area when trying to optimize federated learning.

One potential area to optimize is about what to aggregate, specifically whether to aggregate all parameters or only selected parameters of the model. Another adjustment in this area is whether to aggregate the local models of all workers or only selected workers, if only selected workers then there are many ways to decide how many workers to aggregate and how to select which workers to aggregate.

The next optimization can be regarding when an aggregation should be performed, for instance should an aggregation happen after every epoch of local training on workers, or whether aggregation should be performed only after several epochs of local training on workers. Local training can be made to proceed in strong synchronous manner between all workers, or workers can be allowed to train at different rates and the aggregation schedule and methods would need to be set up according to that.

A third optimization is about where the aggregation should occur, whether it should be done on a single central server, or if it can be done in partial steps on a hierarchy of devices for instance. Finally, the exchange of parameters between local models (on workers) and global models (on parameter servers) can happen in full precision representation, or it can also potentially be in a quantized or reduced bits representation.

Some of these areas of optimization are independent from each other, and thus they can be applied together to be more effective than a single optimization.

3.2 Overview of Selected Optimizations for Federated Learning Proposed in Literature

One optimization strategy for federated learning proposes to categorize workers based on their impact on the global model, differentiating between high-impact and low-impact worker nodes according to the volume and variance of their local datasets [3]. High-impact workers (in the context of classification) are those with high volume of data and many labels, or those with low volume of data and few labels; these workers are assigned a more intensive training and aggregation schedule. Whereas low-impact workers are those with high volume of data but few labels, and these workers are only required to perform local training and be included in the aggregation on every given multiple of training and aggregation rounds by their high-impact counterparts.

Another similar variant of optimization algorithm involves instead calculating a rank for each worker based on how informative that client is (again determined by volume and variety of local data), and using all workers for each aggregation but weighting their contributions to the global model according to the rank [4]. A third proposed approach instead weights the workers for aggregation based on the reliability of calculated ‘soft labels’ of their local dataset [5].

A fourth approach reduces waiting time for aggregation by only considering the earliest x fraction of workers who send their parameter values to the server [6].

All these approaches are able to improve convergence speed without significant impact to the accuracy, and they all result in the need for less network bandwidth than conventional parameter server, and they all are able to outperform FedAvg in experiments reported in the respective papers.

3.2 Edge Parameter Server

3.2.1 Overview

The key concept of EdgePS [7] revolves around the proposition of refraining from aggregating parameters after every local training epoch to expedite the training process. Instead, parameters are aggregated only when further local training cannot enhance the performance of the global model. However, a critical question arises: how do we determine when local training no longer contributes to improving the global model's performance? This question underscores the challenge of optimizing the training process while ensuring the efficacy of the global model. EdgePS is a theoretically inspired algorithm, based on many mathematical derivations.

The first step in EdgePS is to determine whether the distribution of data across workers is homogeneous or heterogeneous, and the algorithm steps for EdgePS differ for each of these two scenarios. The multivariate Kolmogorov-Smirnov test is used to determine this, with each worker computing the statistic for its respective dataset and transmitting the statistic to the server for analysis. However, when the feature space is very high dimensional, workers would directly assume a heterogeneous distribution (as computing the distribution statistic becomes too computationally expensive).

3.2.2 Homogeneous Data Distribution

When the distribution of data across workers is homogeneous, the workers and the parameter server engage in iterative processes until specific termination criteria are met.

Worker

Workers execute the following steps: continuously train their local models until convergence is achieved, then transmit the resultant parameters to the parameter server. Subsequently, they await the reception of aggregated parameters from the server before updating their local models accordingly.

Parameter Server

Meanwhile, the parameter server undertakes its own loop, awaiting parameter submissions from all workers. Upon receipt, it computes aggregated parameters using a specified equation and disseminates them back to all workers for further iteration.

Overall

This iterative exchange between workers and the parameter server facilitates collaborative model refinement in federated learning scenarios, and it continues until some convergence criteria is satisfied for the global model.

3.3.3 Heterogeneous Data Distribution

Similar to the homogeneous case, workers and the server perform iterative loops until termination criteria are met.

Worker

When there is a heterogeneous data distribution, each worker trains its local model for one epoch and communicates its local loss function value to the parameter server (PS), awaiting the receipt of an estimated aggregated loss function value from the PS. The difference from the conventional parameter server (CPS) method here is that in CPS, after every local training epoch, a worker needs to send the server its whole model parameters set; however here, it only needs to send one number (the loss value).

Once the worker receives the estimated aggregated loss function value from the server, the worker uses this number to estimate the lower bound of the loss function value for centralized training (for its own dataset). If this lower bound is less than or equal to zero, the worker sends its model parameters to the PS and awaits further instruction.

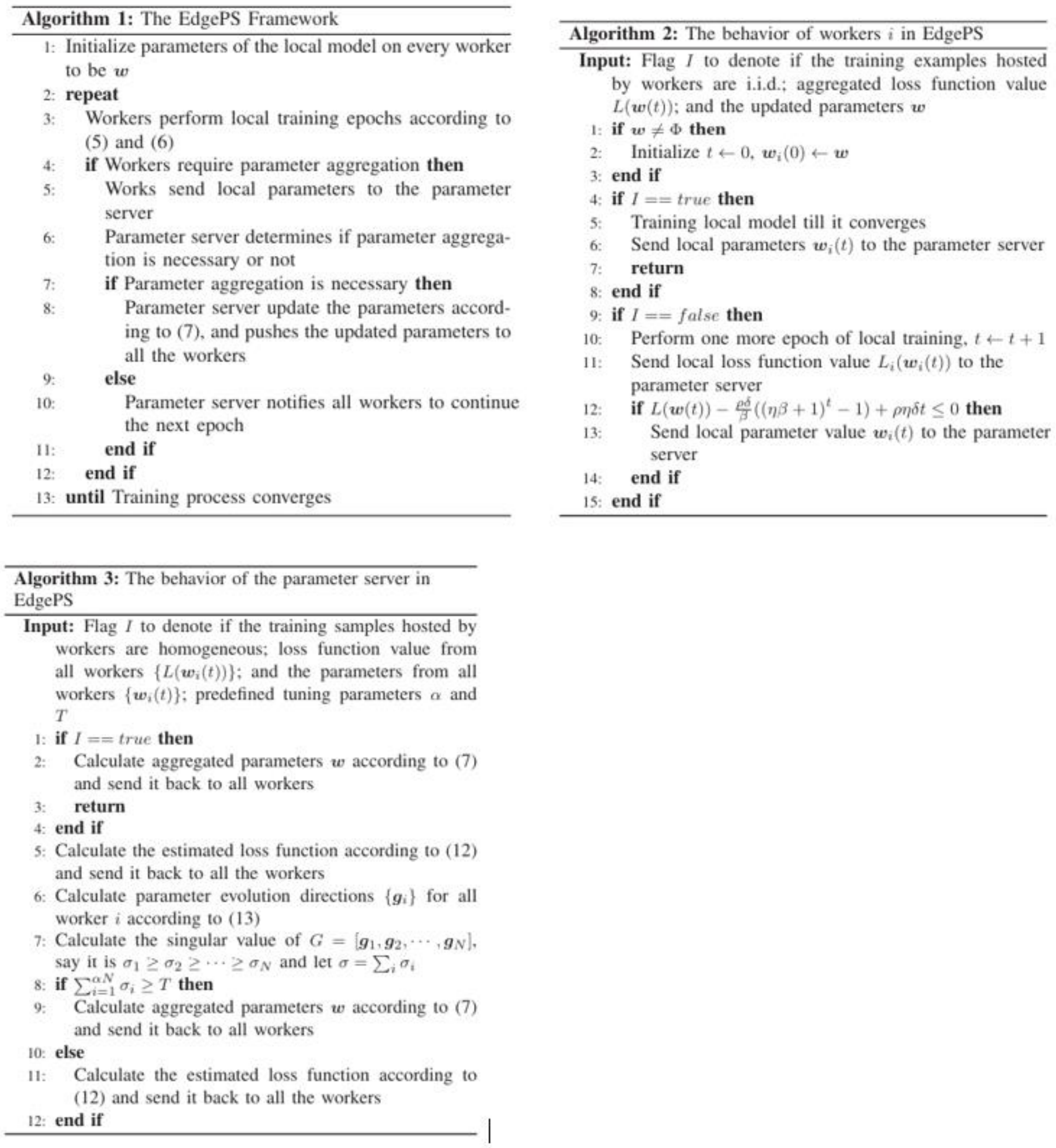
Parameter Server

The PS loop, in turn, waits until all workers' local loss function values are received, then it computes the aggregated loss function and disseminates it to all workers. Once all workers' local model parameters are received, the PS proceeds to calculate parameter evolution directions for all workers, by taking the singular values of the evolution matrix for each worker, and then uses these singular values to calculate a statistic representing the consensus level between workers.

If the statistic surpasses a specified threshold (T), workers are not in alignment with each other for the directions in which they would like to update the parameter, and the PS aggregates parameters and sends them to all workers. Otherwise, the PS does not perform any aggregation and it computes and sends the aggregated loss function to all workers like usual.

Overall

So there are two checkpoints to determine when an aggregation happens: once on the worker side by checking the lower bound of the estimated loss function of centralized training, and once on the parameter server side based on the parameter evolution directions of all workers. This iterative exchange facilitates collaborative model refinement while ensuring efficient communication and coordination between workers and the server in the federated learning environment.



Algorithm 1: The EdgePS Framework

```
1: Initialize parameters of the local model on every worker to be $\boldsymbol{w}$
2: repeat
3:   Workers perform local training epochs according to (5) and (6)
4:   if Workers require parameter aggregation then
5:     Works send local parameters to the parameter server
6:     Parameter server determines if parameter aggregation is necessary or not
7:     if Parameter aggregation is necessary then
8:       Parameter server update the parameters according to (7), and pushes the updated parameters to all the workers
9:     else
10:      Parameter server notifies all workers to continue the next epoch
11:    end if
12:  end if
13: until Training process converges
```

Algorithm 2: The behavior of workers i in EdgePS

```
Input: Flag $I$ to denote if the training examples hosted by workers are i.i.d.; aggregated loss function value $L(\boldsymbol{w}(t))$; and the updated parameters $\boldsymbol{w}$
1: if $w \neq \Phi$ then
2:   Initialize $t \leftarrow 0$, $\boldsymbol{w}_i(0) \leftarrow \boldsymbol{w}$
3: end if
4: if $I == true$ then
5:   Training local model till it converges
6:   Send local parameters $\boldsymbol{w}_i(t)$ to the parameter server
7:   return
8: end if
9: if $I == false$ then
10:  Perform one more epoch of local training, $t \leftarrow t + 1$
11:  Send local loss function value $L_i(\boldsymbol{w}_i(t))$ to the parameter server
12:  if $L(\boldsymbol{w}(t)) - \frac{\rho\delta}{\beta}((\eta\beta + 1)^t - 1) + \rho\eta\delta t \leq 0$ then
13:    Send local parameter value $\boldsymbol{w}_i(t)$ to the parameter server
14:  end if
15: end if
```

Algorithm 3: The behavior of the parameter server in EdgePS

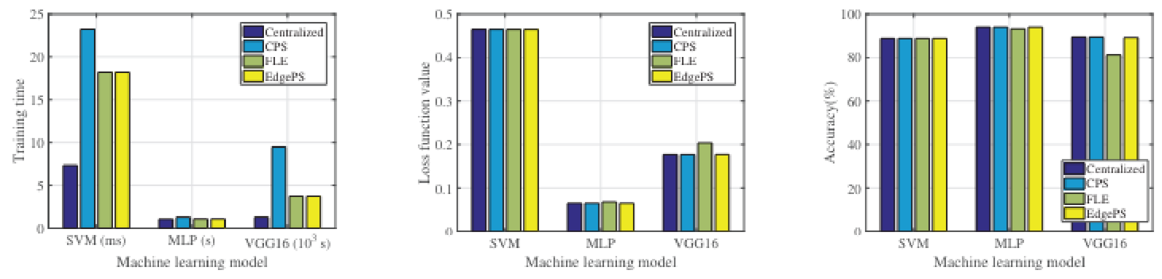
```
Input: Flag $I$ to denote if the training samples hosted by workers are homogeneous; loss function value from all workers $\{L(\boldsymbol{w}_i(t))\}$; and the parameters from all workers $\{\boldsymbol{w}_i(t)\}$; predefined tuning parameters $\alpha$ and $T$
1: if $I == true$ then
2:   Calculate aggregated parameters $\boldsymbol{w}$ according to (7) and send it back to all workers
3:   return
4: end if
5: Calculate the estimated loss function according to (12) and send it back to all the workers
6: Calculate parameter evolution directions $\{\boldsymbol{g}_i\}$ for all worker $i$ according to (13)
7: Calculate the singular value of $G = [\boldsymbol{g}_1, \boldsymbol{g}_2, \cdots, \boldsymbol{g}_N]$, say it is $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N$ and let $\sigma = \sum_i \sigma_i$
8: if $\sum_{i=1}^{\alpha N} \sigma_i \geq T$ then
9:   Calculate aggregated parameters $\boldsymbol{w}$ according to (7) and send it back to all workers
10: else
11:  Calculate the estimated loss function according to (12) and send it back to all the workers
12: end if
```

Figure 1 Algorithm steps for EdgePS, taken from the paper [7]

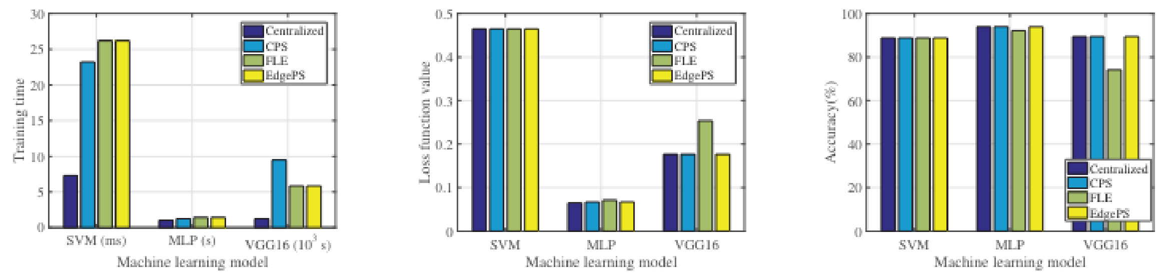
3.3.4 Experiment Results

To test this method, a testbed comprising five servers was utilized, with four servers equipped with GTX 1080 Ti GPUs functioning as workers, while the fifth server operates as the parameter server. Each server maintains a NIC rate of 1Gbps. Across various algorithms tested, including centralized training with data aggregation on a single server, central parameter server with model aggregation after each epoch, EdgePS with variable local epochs before aggregation, and Fixed number of local epochs (FLE), three models—SVM trained on the Default of credit card clients dataset, MLP trained on MNIST, and VGG-16 trained on CIFAR100—are evaluated.

Regardless of the data distribution being homogeneous or heterogeneous, EdgePS consistently demonstrates accuracy and loss function values comparable to both centralized training and conventional distributed training on a central parameter server for all three models. Notably, centralized non-distributed training exhibits the fastest convergence time for the tested models under both homogeneous and heterogeneous data distributions. Among distributed setups, EdgePS and FLE demonstrate faster convergence than central parameter server for all models with homogeneous data. However, in the case of heterogeneous data, central parameter server converges faster than EdgePS/FLE, but this holds only for smaller models, and not for the larger VGG model. Thus to summarize, EdgePS proves to be particularly efficient for large models across all scenarios tested.



(a) Training time with homogeneous training samples (b) Loss function with homogeneous training samples (c) Accuracy with homogeneous training samples



(d) Training time with heterogeneous training samples (e) Loss function with heterogeneous training samples. (f) Accuracy with heterogeneous training samples

Figure 2 Testbed Results for EdgePS, taken from paper [7]

3.4. Resource-based Aggregation Frequency

3.4.1 RAF Steps and Method

Resource-Based Aggregation Frequency (RAF) [8] is a dynamic approach that addresses the straggler issue encountered in federated learning environments by allowing faster workers to train more while slower workers calculate and transmit their updates. RAF varies the aggregation frequency for each worker based on its resource capacity. Coordinating this variation involves assessing the communication and training capacities of each worker and adjusting the number of local training epochs per aggregation at its parent node. This optimization aims to maximize resource utilization and is categorized as weak synchronization, contrasting with strong synchronization methods where all edge devices at the same level have identical aggregation frequencies.

RAF comprises several steps, starting with E-Tree construction based on network topology and data distribution. Next, resource modeling is used to determine aggregation frequencies for each edge device. The process involves computing frequencies from the bottom up and training the model with input frequencies. If the model performs better than a benchmark, fixed aggregation frequencies are outputted. Otherwise, adaptive adjustment of frequencies occurs during training, primarily for leaf nodes, to mitigate large frequency differences among devices that could impede error convergence. RAF employs heuristic algorithms for frequency determination and adaptive communication methods, reducing frequencies with respect to wall clock time to ensure efficient resource utilization across heterogeneous computing environments.

After these preparation steps, the main machine learning model training process adopts a bottom-up approach, with each leaf node performing a designated number of parallel local updates to minimize the local loss function before uploading its model parameters to its parent node. A round of model training concludes when the root node finishes a global aggregation, with results disseminated to its descendants. Model aggregation steps involve non-leaf nodes computing averages from parameters received from child nodes and recursively aggregating them from bottom to top of the tree.

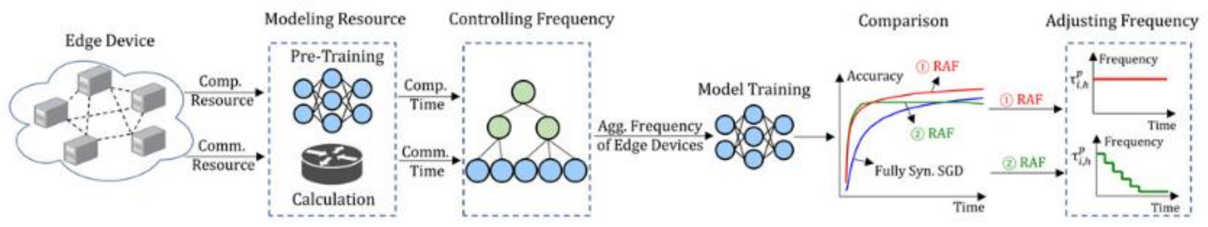


Fig. 4. The Overview Diagram of RAF.

Figure 3 Overview of RAF Process, taken from paper [8]

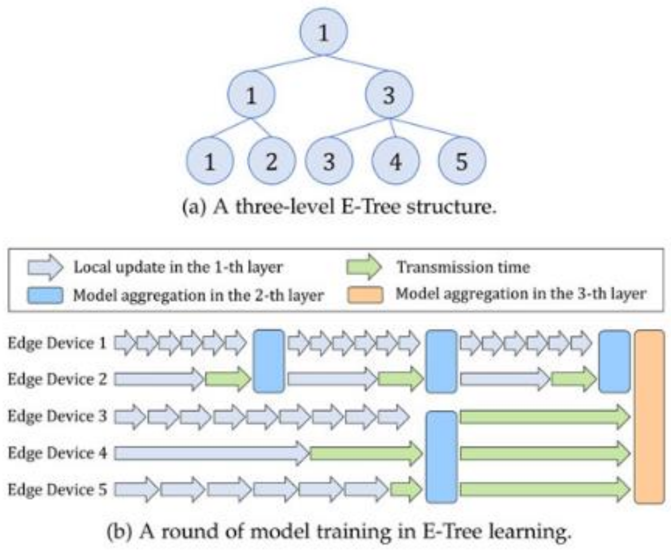

(a) A three-level E-Tree structure.
Local update in the 1-th layer
Transmission time
Model aggregation in the 2-th layer
Model aggregation in the 3-th layer
Edge Device 1
Edge Device 2
Edge Device 3
Edge Device 4
Edge Device 5
(b) A round of model training in E-Tree learning.

Figure 4 Illustration of example of varying number of local epochs of training for different nodes at the same level, taken from paper [8]

3.4.2 Experiments and Results

A testbed was created by the authors and used to experiment on logistic regression (LR) and a 2 layer multilayer perceptron (MLP) trained on MNIST, and convolutional neural network (CNN) trained on MNIST, Fashion-MNIST, and EMNIST. Three methods were tried for the federated training: RAF, fully synchronous, and hierarchical federated learning (HFL).

The experiments were run for both IID and non-IID distributions of data across workers, and it was found that RAF achieved much better accuracy compared to fully synchronous and HFL if training time is constraint under all non-IID levels.

When examining the rate of convergence, RAF demonstrates a quicker attainment of a designated training accuracy compared to both fullySyn and HFL. Also, the accuracy of RAF notably improves by up to 3.99%, 6.67%, and 10.67% respectively when training time is limited, under scenarios involving 20, 50, and 100 edge devices, in contrast to fullySyn. These findings indicate that as the number of edge devices escalates, the disparity in convergence accuracy between RAF and fullySyn widens.

The advantages of RAF over fully synchronous training were found to continue to hold even for different model structures.

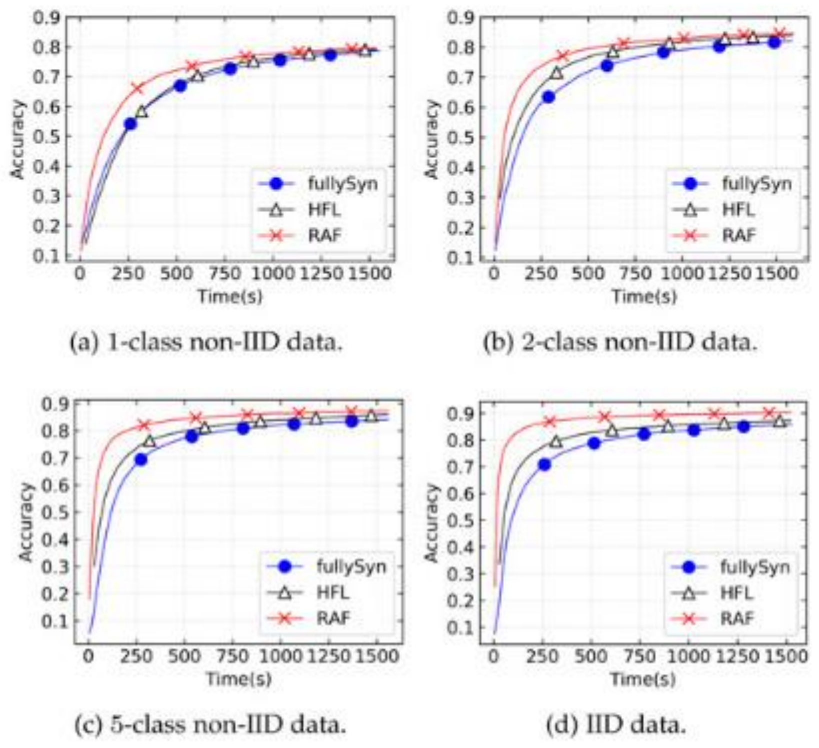
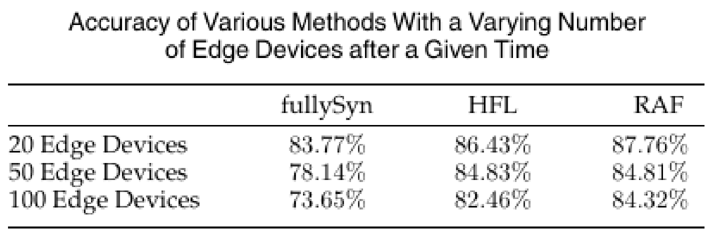


Figure 5 Comparison of convergence on LR model on MNIST, for the 3 methods; figure taken from paper [8]



Accuracy of Various Methods With a Varying Number of Edge Devices after a Given Time

	fullySyn	HFL	RAF
20 Edge Devices	83.77%	86.43%	87.76%
50 Edge Devices	78.14%	84.83%	84.81%
100 Edge Devices	73.65%	82.46%	84.32%

Figure 6 Taken from paper [8]; description in figure header

4. Discussion and Analysis

RAF requires a hierarchical tree setup for the devices, whereas EdgePS operates with a central parameter server configuration. EdgePS dynamically determines the number of local training epochs each worker should conduct before aggregation by using a statistic to define a point beyond which additional local training is not beneficial without aggregation. However, EdgePS can only aggregate when every worker, including the slowest one, has reached this point, causing faster workers to idle until then.

In contrast, RAF varies the aggregation frequency for each worker based on the worker's communication and computation resource capacities, and RAF allows different parts of the tree to aggregate in parallel.

For a given aggregation, one worker might have completed five epochs and be far from convergence, while another worker under the same aggregator could have completed ten epochs and be closer to convergence. These two workers' local models would be aggregated together, preventing faster workers from idling.

The main advantage of EdgePS is that after each epoch of local training, only one loss value number needs to be sent to the parameter server, instead of all model parameters from every worker, which saves a lot of communication cost. However, EdgePS is not able to overcome the issue of stragglers. RAF effectively addresses especially the straggler issue, and partially also reduces the communication load (compared to conventional parameter server method) as workers train multiple epochs locally before sending parameters to their aggregator node.

Both EdgePS and RAF do require all workers on a given level to have completed their local training before aggregating. However as mentioned earlier in the paper, research has shown feasibility for alternative approaches where not all workers would be included in every aggregation, with the criteria for inclusion / exclusion being related to the speed of the worker or the distribution of local data on that particular worker.

Both EdgePS and RAF can be enhanced with strategies such as quantization and pruning to further improve their efficiency and effectiveness.

4. References

[1] M. Li, X. Tang, S. Chen, Y. Weng, L. Peng and W. Yang, "Exploring the Impact of Non-IID on Federated Learning," in *International Conference on Blockchain Technology and Information Security (ICBCTIS)*, 2023.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks," *arXiv,* 2023.

[3] D. M. S. Bhatti, M. Haris and H. Nam, "A Communication Efficient Approach of Global Training in Federated Learning," in *13th International Conference on Information and Communication Technology Convergence (ICTC)* , 2022.

[4] D. M. S. Bhatti and H. Nam, "A Performance Efficient Approach of Global Training in Federated Learning," in *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2023.

[5] Y. Yang, Y. Luo and G. Zhu, "An optimized federated learning method based on soft label grouping for heterogeneous IoT," *Cluster Computing (Springer),* 2024.

[6] A. Kulkarni, A. Kumar, R. Shorey and R. Verma, "Towards an Efficient Federated Learning Framework with Selective Aggregation," in *16th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, 2024.

[7] Y. Zhao, Y. Hou and C. Qiao, "EdgePS: Selective Parameter Aggregation for Distributed Machine Learning in Edge Computing," in *IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021.

[8] L. Yang, Y. Gan, J. Cao and Z. Wang, "Optimizing Aggregation Frequency for Hierarchical Model Training in Heterogeneous Edge Computing," *IEEE TRANSACTIONS ON MOBILE COMPUTING,* vol. 22, no. 7, pp. 4181-4194, 2023.