Federated Learning

Final project report

By

Rohith Lingala
Manish Valeti
Sathwik Busireddy
Navyanth Bollareddy

Guided by-

Dr. Jin Lu
CSCI 8000- Adv Special Topics

Dec 09th, 2024

Introduction

Federated Learning (FL) has emerged as a powerful machine learning paradigm that enables collaborative model training across decentralized data sources while maintaining data privacy. This unique approach addresses the challenges posed by data silos and privacy concerns, making it particularly suitable for applications with strict privacy requirements, such as healthcare, finance, and IoT systems. However, FL is not without its challenges. The heterogeneity of data across clients, communication overhead, and the need for personalization are some of the bottlenecks that affect the performance and scalability of federated learning systems.

Among the critical challenges in FL, data heterogeneity remains a primary focus. Personalized Federated Learning (PFL) has emerged as a solution, allowing models to adapt to individual client distributions while retaining the benefits of federated training. One notable approach is **Personalized Federated Learning with Feature Alignment and Classifier Collaboration**, which emphasizes leveraging shared feature representations and customized classifier heads for better local-global alignment. By incorporating global semantic knowledge into local representation learning and enabling fine-grained collaboration between classifier heads, this method addresses the limitations of prior approaches in terms of generalization and personalization.

Additionally, other research directions in FL include **Agnostic Federated Learning**, which aims to optimize models for any target distribution formed by mixtures of client distributions, promoting fairness across clients, and **Meta-Learning for Personalized Federated Learning**, which leverages Model-Agnostic Meta-Learning (MAML) to enable quick adaptation of shared models to new clients. These frameworks collectively provide a foundation for addressing the inherent complexities of federated learning in diverse settings.Whereas our project focuses on the advancements proposed in the paper, Personalized Federated Learning with Feature Alignment and Classifier Collaboration, which offers an innovative approach to overcoming data heterogeneity and improving model generalization in FL. The methodology and its effectiveness in heterogeneous data scenarios are analyzed to understand its impact on federated learning systems.

Motivation

Federated Learning (FL) is a distributed machine learning paradigm designed to collaboratively train models across multiple decentralized clients, such as mobile devices or organizations, without requiring the transfer of raw data. This ensures data privacy and security, making FL particularly appealing for sensitive applications like healthcare, finance, and IoT systems. Instead of sharing data, clients exchange model updates (e.g., gradients or weights), which are aggregated at a central server to update a global model. This distributed approach mitigates privacy risks but introduces challenges such as data heterogeneity, communication overhead, and personalization needs.

Introduction to FL Algorithms:

1. Federated Averaging(FedAVG) :

FedAVG is the foundational algorithm for FL, combining local stochastic gradient descent (SGD) with a centralized aggregation mechanism. Each client trains its local model using its data and then sends the updated model weights to a central server. The server computes a weighted average of these updates to form the global model.

Mathematical Formulation: The global update is computed as:

$$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1}$$

where:

- w^{t+1}: Global model weights at round $t+1$,
- K: Number of clients,
- n_k: Number of data points on client k,
- n: Total number of data points across all clients,
- w_k^{t+1}: Local model weights of client k after round t.

FedAVG is computationally efficient but struggles with non-IID data, where client data distributions differ significantly.

2. Layered Global FedAVG(LG_FedAVG):

LG_FedAVG extends FedAVG by introducing layered-global fine-tuning. After training the global model, clients can adapt it to their local data by fine-tuning specific layers, enabling personalized model performance.

Mathematical Concept: The optimization problem becomes:

$$\min_{w} \sum_{k=1}^{K} p_k F_k(w) + \lambda ||w - w_k||^2$$

where:

- $F_k(w)$: Loss function of client k,
- p_k: Proportion of data on client k,
- w: Global model weights,
- w_k: Local model weights,
- λ: Regularization term for personalization.

3. Personalized Federated(FedPer):

FedPer introduces model personalization by splitting the model into shared and client-specific components. The shared component is trained collaboratively across clients, while the personalized layers are updated independently on each client.

Mathematical Formulation: Let w_s represent the shared weights and w_p^k the personalized weights for client k. The optimization problem can be written as:

$$\min_{w_s} \sum_{k=1}^{K} F_k(w_s, w_p^k)$$

where Fk is the loss function for client kk. This approach ensures that the shared representation captures global features, while personalization layers adapt to local data.

4. Local Training:

Local training serves as a baseline where each client trains its model independently without any collaboration. While this avoids communication overhead, it does not leverage shared knowledge across clients, leading to isolated models with potentially limited generalization.

Mathematical Formulation: For client k, the objective is:

$$\min_{w_k} F_k(w_k)$$

5. FedPAC:

FedPAC addresses data heterogeneity by leveraging both global and local feature representations. It aligns the local and global feature spaces using a shared encoder and promotes collaboration among local classifier heads through weighted combinations.

1. **Feature Alignment:** Local feature alignment is performed by minimizing

$$\min_{w_e} \sum_{k=1}^{K} \| f_k(x; w_e) - f_g(x; w_e) \|^2$$

where fk(x;we) and fg(x;we) represent the local and global feature representations, respectively.

2. **Classifier Collaboration:** Classifier combination weights are optimized as:

$$\min_{w_c} \sum_{k=1}^{K} \mathcal{L}(h_k(x; w_c), y)$$

where hk(x;wc) is the combined classifier for client k, and L is the loss.

Main Idea :

The project focuses on comparing the performance of various Federated Learning (FL) algorithms: **FedAVG**, **LG_FedAVG**, **FedPer**, **Local**, and **FedPAC**. The primary goal is to analyze their effectiveness under different scenarios, specifically addressing challenges like data heterogeneity and personalization. Below, I provide a detailed explanation of the origin and concept of each algorithm, along with pseudocode. This idea is from one of the papers which presented in the class i.e., **Personalized Federated Learning with Feature Alignment and Classifier Collaboration.**

1. **FedAVG :** The provided code implements the FedAvg algorithm, a fundamental technique in Federated Learning (FL) for distributed model training. In each communication round, the server selects a subset of clients to participate, based on a predefined fraction of users. These clients receive the current global model weights and perform local training for a specified number of epochs. After

```python
# vanila FedAvg
def train_round_fedavg(args, global_model, local_clients, rnd, **kwargs):
    print(f'\n---- Global Communication Round : {rnd+1} ----')
    num_users = args.num_users
    m = max(int(args.frac * num_users), 1)
    if (rnd >= args.epochs):
        m = num_users
    idx_users = np.random.choice(range(num_users), m, replace=False)
    idx_users = sorted(idx_users)
    local_weights, local_losses1, local_losses2 = [], [], []
    local_grads = []
    local_acc1 = []
    local_acc2 = []
    agg_weight = []

    global_weight = global_model.state_dict()

    for idx in idx_users:
        local_client = local_clients[idx]
        agg_weight.append(local_client.agg_weight)
        local_epoch = args.local_epoch
        local_client.update_local_model(global_weight=global_weight)
        w, loss1, loss2, acc1, acc2 = local_client.local_training(local_epoch=local_epoch, round=rnd)
        local_weights.append(copy.deepcopy(w))
        local_losses1.append(copy.deepcopy(loss1))
        local_losses2.append(copy.deepcopy(loss2))
        local_acc1.append(acc1)
        local_acc2.append(acc2)

    # get global weights
    global_weight = average_weights_weighted(local_weights, agg_weight)
    # update global model
    global_model.load_state_dict(global_weight)

    loss_avg1 = sum(local_losses1) / len(local_losses1)
    loss_avg2 = sum(local_losses2) / len(local_losses2)
    acc_avg1 = sum(local_acc1) / len(local_acc1)
    acc_avg2 = sum(local_acc2) / len(local_acc2)

    return loss_avg1, loss_avg2, acc_avg1, acc_avg2
```

training, clients return their updated model weights, along with metrics like local losses and accuracies, to the server. The server aggregates these updates using a weighted average, where weights are proportional to the data sizes of the participating clients. The global model is then updated with the aggregated weights, and performance metrics, such as average loss and accuracy, are computed to evaluate the training progress. This approach ensures scalable, privacy-preserving learning while addressing the challenges of decentralized data and heterogeneous client participation.

2. **LG_FedAvg :** The LG_FedAVG (Layered-Global Federated Averaging) algorithm, which extends the traditional FedAVG approach by incorporating personalization through parameter decoupling. In each communication round, a subset of clients is randomly selected to participate in the training. These clients receive the global model weights and perform local updates. Each client

```python
# parameter decoupling
def train_round_lgfedavg(args, global_model, local_clients, rnd, **kwargs):
    print(f'\n---- Global Communication Round : {rnd+1} ----')
    num_users = args.num_users
    m = max(int(args.frac * num_users), 1)
    if (rnd >= args.epochs):
        m = num_users
    idx_users = np.random.choice(range(num_users), m, replace=False)
    idx_users = sorted(idx_users)
    local_weights, local_losses1, local_losses2 = [], [], []
    local_grads = []
    local_acc1 = []
    local_acc2 = []
    agg_weight = []

    global_weight = global_model.state_dict()

    for idx in idx_users:
        local_client = local_clients[idx]
        agg_weight.append(local_client.agg_weight)
        local_epoch = args.local_epoch
        local_client.update_local_model(global_weight=global_weight)
        w, loss1, loss2, acc1, acc2 = local_client.local_training(local_epoch=local_epoch)
        local_weights.append(copy.deepcopy(w))
        local_losses1.append(copy.deepcopy(loss1))
        local_losses2.append(copy.deepcopy(loss2))
        local_acc1.append(acc1)
        local_acc2.append(acc2)

    # get global weights
    global_weight = average_weights_weighted(local_weights, agg_weight)
    # global_weight = average_weights(local_weights)
    # update global model
    global_model.load_state_dict(global_weight)

    loss_avg1 = sum(local_losses1) / len(local_losses1)
    loss_avg2 = sum(local_losses2) / len(local_losses2)
    acc_avg1 = sum(local_acc1) / len(local_acc1)
    acc_avg2 = sum(local_acc2) / len(local_acc2)

    return loss_avg1, loss_avg2, acc_avg1, acc_avg2
```

personalizes their local model by fine-tuning it for a specified number of epochs while updating its local parameters based on its unique dataset. The server then collects the updated local weights and aggregates them using a weighted averaging approach, considering the data size of each client. The global model is updated with the aggregated weights, ensuring improved performance across all clients. Additionally, metrics such as local losses and accuracies are averaged to evaluate model performance. This algorithm balances global generalization and local personalization, making it suitable for heterogeneous data environments.

3. **Fedper :** FedPer (Federated Personalization) algorithm, designed to balance shared global knowledge with local personalization in federated learning. In this approach, the server selects a subset of clients for each communication round

```python
def train_round_fedper(args, global_model, local_clients, rnd, **kwargs):
    print(f'\n---- Global Communication Round : {rnd+1} ----')
    num_users = args.num_users
    m = max(int(args.frac * num_users), 1)
    if (rnd >= args.epochs):
        m = num_users
    idx_users = np.random.choice(range(num_users), m, replace=False)
    idx_users = sorted(idx_users)
    local_weights, local_losses1, local_losses2 = [], [], []
    local_grads = []
    local_acc1 = []
    local_acc2 = []
    agg_weight = []

    global_weight = global_model.state_dict()

    for idx in idx_users:
        local_client = local_clients[idx]
        local_epoch = args.local_epoch
        agg_weight.append(local_client.agg_weight)
        local_client.update_local_model(global_weight=global_weight)
        w, loss1, loss2, acc1, acc2 = local_client.local_training(local_epoch=local_epoch, round=rnd)
        local_weights.append(copy.deepcopy(w))
        local_losses1.append(copy.deepcopy(loss1))
        local_losses2.append(copy.deepcopy(loss2))
        local_acc1.append(acc1)
        local_acc2.append(acc2)

    # get global weights
    global_weight = average_weights_weighted(local_weights, agg_weight)
    # update global model
    global_model.load_state_dict(global_weight)

    loss_avg1 = sum(local_losses1) / len(local_losses1)
    loss_avg2 = sum(local_losses2) / len(local_losses2)
    acc_avg1 = sum(local_acc1) / len(local_acc1)
    acc_avg2 = sum(local_acc2) / len(local_acc2)

    return loss_avg1, loss_avg2, acc_avg1, acc_avg2
```

and broadcasts the global model weights. Each participating client then updates its local model by training both global shared layers and its client-specific personalized layers using its local data. After training, the updated local model weights, along with corresponding metrics like local losses and accuracies, are sent back to the server. The server aggregates the shared global weights using a weighted average approach, considering each client's data contribution. The personalized layers remain unique to each client, allowing the model to adapt better to local data distributions. The function computes average metrics such as losses and accuracies across the participating clients to monitor model performance. This algorithm effectively handles data heterogeneity by leveraging both global and client-specific knowledge, enhancing the personalization capabilities of federated learning systems.

4. **Local :** It is a baseline approach in federated learning where each client trains its model independently without any global model aggregation or collaboration. In this method, all participating clients train their local models using only their data for a specified number of epochs. The global model is initialized but not updated or used for aggregation since there is no communication between the clients and

```python
# local training only
def train_round_standalone(args, global_model, local_clients, rnd, **kwargs):
    print(f'\n---- Global Communication Round : {rnd+1} ----')
    num_users = args.num_users
    m = max(int(args.frac * num_users), 1)
    if (rnd >= args.epochs):
        m = num_users
    idx_users = np.random.choice(range(num_users), m, replace=False)
    idx_users = sorted(idx_users)
    local_losses1, local_losses2 = [], []#, []
    local_acc1 = []
    local_acc2 = []

    global_weight = global_model.state_dict()

    for idx in idx_users:
        local_client = local_clients[idx]
        local_epoch = args.local_epoch
        w, loss1, loss2, acc1, acc2 = local_client.local_training(local_epoch=local_epoch)
        local_losses1.append(copy.deepcopy(loss1))
        local_losses2.append(copy.deepcopy(loss2))
        local_acc1.append(acc1)
        local_acc2.append(acc2)

    loss_avg1 = sum(local_losses1) / len(local_losses1)
    loss_avg2 = sum(local_losses2) / len(local_losses2)
    acc_avg1 = sum(local_acc1) / len(local_acc1)
    acc_avg2 = sum(local_acc2) / len(local_acc2)

    return loss_avg1, loss_avg2, acc_avg1, acc_avg2
```

the server. Each client computes its local losses and accuracies during training, which are logged for evaluation purposes. The function calculates and returns average metrics, such as losses and accuracies, across all clients to assess the overall performance of the locally trained models. This approach eliminates communication overhead but does not leverage shared knowledge from other clients, often resulting in suboptimal performance when compared to collaborative federated learning algorithms. It serves as a baseline to evaluate the effectiveness of other FL methods.

5. **FedPAC :** The FedPAC (Personalized Federated Learning with Feature Alignment and Classifier Collaboration) algorithm addresses the challenges of data heterogeneity in federated learning by combining global feature alignment with personalized classifier collaboration. In each communication round, the server selects a subset of clients, broadcasts the global model weights and centroids, and collects locally updated models and feature statistics. The server updates the global feature extractor and optimizes personalized classifiers for each client, ensuring both global consistency and local adaptability. On the client

Algorithm 1 `FedPAC`

Input: Learning rate $\{\eta_f, \eta_g\}$, number of communication rounds T, number of clients m, number of local epochs E, hyper-parameter λ
Server Executes:
Initialize:$\boldsymbol{w}^{(0)}$ and $\boldsymbol{c}^{(0)}$
for $t = 0, 1, ..., T-1$ **do**
  Select client set C_t
  Broadcast $\{\boldsymbol{w}^{(t)}, \boldsymbol{c}^{(t)}\}$ to selected clients for local update
  Collect models and statistics from clients
  Get global feature extractor $\boldsymbol{\theta}^{(t+1)}$ as (18) and global centriods $\boldsymbol{c}^{(t+1)}$ as (19)
  Get personalized classifier $\tilde{\boldsymbol{\phi}}_i^{(t+1)}$ by solving (11)
  Send $\{\boldsymbol{\theta}^{(t+1)}, \tilde{\boldsymbol{\phi}}_i^{(t+1)}\}$ back to client $i \in C_t$
end for
ClientUpdate(i, $\boldsymbol{w}^{(t)}$, $\boldsymbol{c}^{(t)}$):
  Update local feature extractor $\boldsymbol{\theta}_i^{(t)} \leftarrow \tilde{\boldsymbol{\theta}}^{(t)}$
  Extract feature statistics $\boldsymbol{\mu}_i^{(t)}$ and $V_i^{(t)}$
  Train $\boldsymbol{\phi}_i^{(t+1)}$ for 1 epoch and $\boldsymbol{\theta}_i^{(t+1)}$ for E epoch by turns
  Extract local feature centroids $\hat{\boldsymbol{c}}_i^{(t+1)}$ as (17)
  Return $\{\boldsymbol{\theta}_i^{(t+1)}, \boldsymbol{\phi}_i^{(t+1)}, \hat{\boldsymbol{c}}_i^{(t+1)}, \boldsymbol{\mu}_i^{(t)}, V_i^{(t)}\}$ to server

side, feature alignment is achieved by updating the local feature extractor based on the global model, while the classifier and extractor are alternately trained on local data for a specified number of epochs. Additionally, clients compute feature centroids and extract statistics to assist in global model updates. By integrating global knowledge with local personalization, FedPAC effectively balances model generalization and specialization, making it well-suited for heterogeneous and decentralized data environments.

Datasets:

The datasets used for this project focus on image classification tasks and include four popular benchmarks: **EMNIST, Fashion-MNIST, CIFAR-10, and CINIC-10**, each selected for its unique characteristics and challenges. EMNIST is a 62-class dataset that extends the classic MNIST by including 10 digits, 26 uppercase letters, and 26 lowercase letters, making it suitable for diverse handwritten character recognition tasks. Fashion-MNIST consists of 10 categories of clothing, offering a modern alternative to MNIST with a focus on more complex and visually relevant categories. CIFAR-10 includes 10 categories of color images, representing various objects and animals, and provides a standard benchmark for evaluating image classification models. CINIC-10, a composite dataset derived from CIFAR-10 and ImageNet, is more diverse and challenging, testing the scalability and robustness of algorithms in handling large-scale and heterogeneous data.

For these datasets, two Convolutional Neural Network (CNN) models are employed. The first model consists of two convolutional layers with 16 and 32 channels, each followed by a max-pooling layer, and two fully connected layers with 128 and 10 units before the softmax output. The second model, designed for more complex datasets like CIFAR-10 and CINIC-10, includes an additional convolutional layer with 64 channels. Both models use the LeakyReLU activation function for better gradient flow. To simulate realistic federated learning scenarios, data partitioning strategies are applied to introduce heterogeneity. Each client receives data with a mix of dominant and uniformly sampled classes, ensuring a controlled degree of non-IID distribution. For

Fashion-MNIST, CIFAR-10, and CINIC-10 (10-class datasets), clients are divided into five groups, each focusing on three consecutive dominant classes. Similarly, for EMNIST, clients are divided into three groups based on digits, uppercase letters, and lowercase letters. This partitioning mimics real-world data distributions, enabling a meaningful evaluation of federated learning algorithms in handling non-IID data.

Implementation :

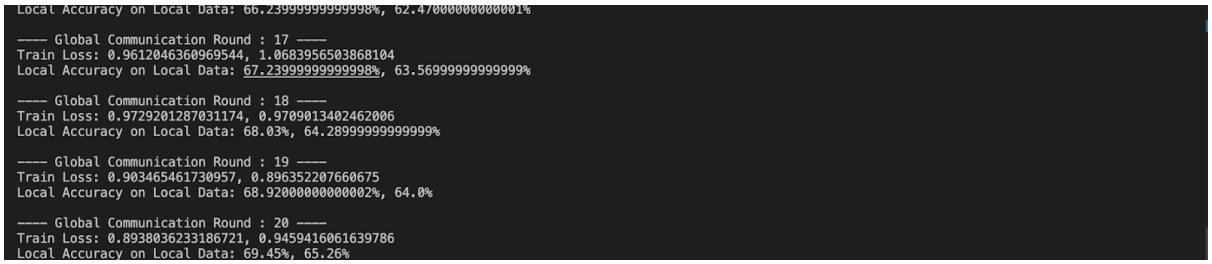
In our implementation, we utilized mini-batch Stochastic Gradient Descent (SGD) as the local optimizer across all federated learning approaches. For EMNIST and Fashion-MNIST datasets, the local training step size (η) was set to 0.01, while for CIFAR-10 and CINIC-10, it was increased to 0.02. To reduce computational overhead, the classifier was trained for one epoch with a larger step size ($\eta g=0.1$), while the feature extractor was trained for multiple epochs using the same step size ($\eta f=\eta$) as the baselines. Other hyperparameters included a weight decay of $5\times10-4$ and a momentum of 0.5.

The batch size was fixed at 50 for all datasets except EMNIST, where it was set to 100. Local training was conducted over five epochs ($E=5$) per communication round, with a total of 200 global communication rounds for all datasets. This configuration ensured that the federated learning approaches reached convergence with minimal or no further accuracy improvements beyond the set rounds. For performance evaluation, we reported the average test accuracy across all local models to provide a comprehensive comparison of methods. These settings ensured consistent and fair evaluations of the implemented algorithms under various data distributions and conditions.

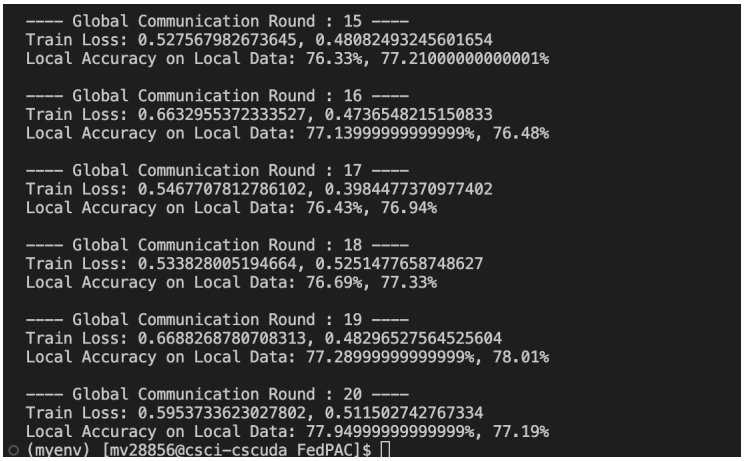
Results :

Performance comparison of five Federated Learning methods—**FedPAC**, **FedAvg**, **LG-FedAvg**, **FedPer**, and **Local Training**—across four datasets (EMNIST, Fashion-MNIST, CIFAR-10, and CINIC-10) with two configurations (20 clients and 100 clients). The results are expressed as the average test accuracy of the local models, providing insights into the effectiveness of each method under varying data and client distributions.

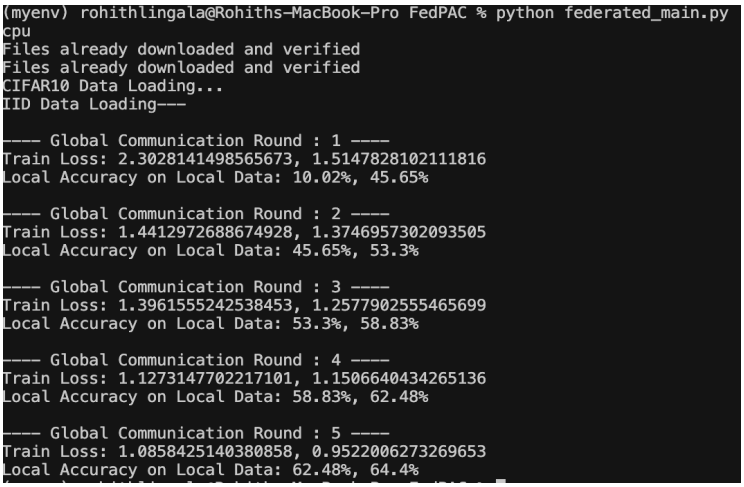
Below are some of the snapshots taken at the time of execution and the programs run through hours for 200 epochs with a variation of 20 and 100 clients on the same dataset.



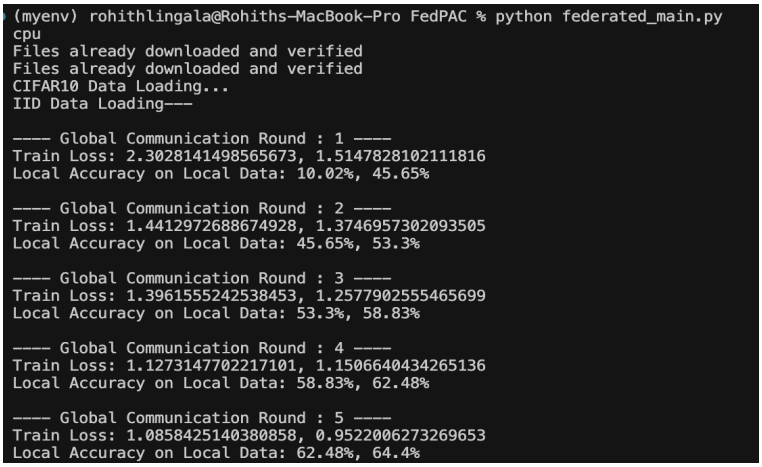
FedAVG execution



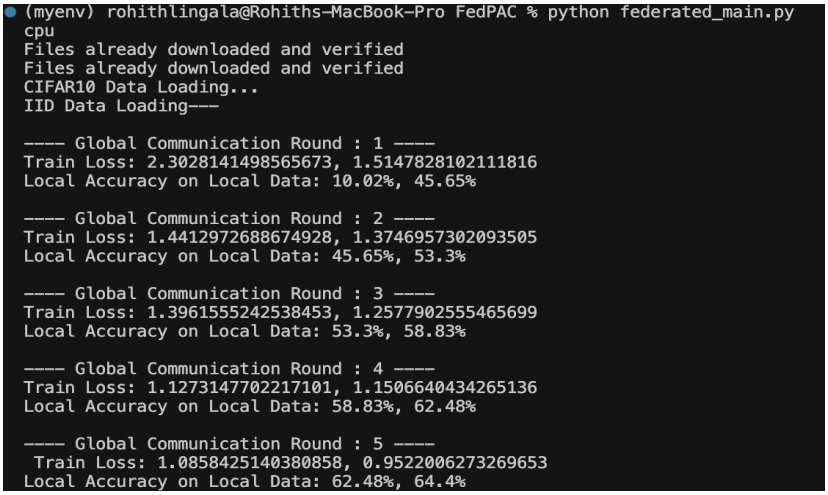
LG_FedAVG



LOCAL Execution



FedPerExecution



FedPAC Exection

FedPAC consistently outperforms other methods across all datasets and client configurations, demonstrating its ability to effectively handle non-IID data and provide robust personalization. For instance, on EMNIST, FedPAC achieves 86.46% and 89.88% accuracy for 20 and 100 clients, respectively, which is significantly higher than FedAvg (71.85% and 75.97%) and LG-FedAvg (74.76% and 75.81%). Similarly, for Fashion-MNIST, FedPAC reaches accuracies of 91.83% (20 clients) and 92.72% (100 clients), outperforming FedPer and Local, which achieve slightly lower scores.

Dataset	EMNIST		Fashion-MNIST		CIFAR-10		CINIC-10	
Method	20 clients	100 clients	20 clients	100 clients	20 clients	100 clients	20 clients	100 clients
FedPAC	86.46	89.88	91.83	92.72	81.13	83.36	74.96	77.36
FedAvg	71.85	75.97	85.28	86.89	70.05	73.82	58.38	62.52
LG-FedAvg	74.76	75.81	85.66	86.28	65.19	65.38	63.75	63.93
FedPer	75.91	77.06	87.43	87.88	68.37	72.26	63.47	66.26
Local	73.85	74.11	85.68	86.37	65.43	64.68	63.19	63.33

For more complex datasets like CIFAR-10 and CINIC-10, FedPAC maintains its superior performance, achieving 81.13% and 83.36% on CIFAR-10 and 74.96% and 77.36% on CINIC-10 for 20 and 100 clients, respectively. Other methods, such as FedAvg and LG-FedAvg, show lower performance, particularly on these datasets, with accuracies ranging between 58.38% and 73.82%. While FedPer and Local methods exhibit better performance than FedAvg on certain datasets, they consistently fall short of FedPAC.

Overall, the results highlight the superiority of FedPAC in achieving high accuracy, especially in heterogeneous and large-scale federated learning settings, due to its combination of global feature alignment and personalized classifier collaboration. This demonstrates its effectiveness in balancing generalization and personalization compared to traditional and baseline methods.

Conclusion :

This project demonstrates the effectiveness of different Federated Learning algorithms, including FedPAC, FedAvg, LG-FedAvg, FedPer, and Local Training, across diverse datasets and client configurations. Among these, FedPAC consistently outperformed other methods, achieving the highest accuracy by effectively addressing data heterogeneity through global feature alignment and personalized classifier collaboration. The results highlight the importance of balancing generalization and personalization in federated learning, especially in non-IID and large-scale data settings. This comprehensive comparison provides valuable insights into the trade-offs of different FL approaches, paving the way for future advancements in privacy-preserving and distributed learning systems.

References:

[1] Xu, Jian, Xinyi Tong, and Shao-Lun Huang. "Personalized federated learning with feature alignment and classifier collaboration." arXiv preprint arXiv:2306.11867 (2023).

[2] Wen, Jie, et al. "A survey on federated learning: challenges and applications." International Journal of Machine Learning and Cybernetics 14.2 (2023): 513-535.

[3] Mohri, Mehryar, Gary Sivek, and Ananda Theertha Suresh. "Agnostic federated learning." International conference on machine learning. PMLR, 2019.

[4] Personalized Federated Learning: A Meta-Learning Approach Alireza Fallah∗, Aryan Mokhtari†, Asuman Ozdaglar.

Contributions:

Num	Paper Title	Presenter
1	A survey on federated learning: challenges and applications.	Manish Valeti
2	Personalized federated learning with feature alignment and classifier collaboration.	Rohith Lingala
3	Agnostic federated learning.	Sathwik Reddy
4	Personalized Federated Learning: A Meta-Learning Approach	Navyanth Reddy

Project Contributions:

Work has been distributed equally among all the team members and each person worked on different dataset by training and testing all the above discussed models.

- Rohith Lingala - Worked on **EMNIST** dataset for all the models
- Manish Valeti - Worked on **Fashion-MNIST** for all the models
- Sathwik - Worked on **CIFAR-10** for all the models
- Navyanth - Worked on **CINIC-10** for all the models