Delft University of Technology
Master of Science Thesis in Embedded Systems

Federated Learning with Rebalanced Dataset

Tianyi Liu

TU Delft Delft University of Technology

Federated Learning with Rebalanced Dataset

Master of Science Thesis in Embedded Systems

Embedded Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Tianyi Liu

November 25 2023

Author
Tianyi Liu (T.Liu-14@student.tudelft.com)
(ltylty221@gmail.com)
Title
Federated Learning with Rebalanced Dataset
MSc Presentation Date
November 30 2023

Graduation Committee

Dr. Qing Wang	Delft University of Technology
Dr. Jie Yang	Delft University of Technology
MingKun Yang	Delft University of Technology

Abstract

With the widespread application of artificial intelligence, centralized machine learning approaches, which require access to users' local data, have raised concerns about data privacy. In response, federated learning, an architecture that aggregates models trained locally with local data, has been proposed. This approach addresses the data privacy issues inherent in centralized machine learning while also alleviating the high communication costs and server resource demands. However, due to its architectural nature, federated learning, with its single global model, struggles to meet the diverse personalized needs of clients and suffers significant accuracy degradation when client data distributions are uneven or exhibit non-IID characteristics. Personalized federated learning has been introduced to address these issues of data heterogeneity and personalized needs. Its goal is not to train a single global model but to ensure that each client participating in the personalized federated learning framework has a local model that meets their individual needs. Yet, personalized federated learning also has its shortcomings: the global model in federated learning often becomes an intermediary product in this framework, lacking the advantage of learning a generalized model. This thesis proposes a new personalized federated learning scheme, Federated Learning with Rebalanced Dataset(FedReb), which is based on parameter decoupling. By introducing a rebalanced dataset generated according to the distribution of clients' local data, this framework achieves high accuracy for both the global model and average client models. Comparative experiments demonstrate its superior scalability and robustness over other federated learning and personalized federated learning algorithms, and the report suggests optimal configurations for achieving the best results with reasonable costs. Additionally, a testbed has been established, and the deployment of the algorithm on it has been realized, verifying the feasibility of the algorithm in real-world setups.

Preface

My interest in federated learning was sparked during the summer of 2022, when I assisted my daily supervisor, MingKun Yang, and Ran Zhu in building a federated learning testbed using Raspberry Pi. My previous experience with Raspberry Pi and a keen interest in hardware-related projects led me to this opportunity. It was through this project that I first encountered the concept of federated learning and its challenges in edge computing. This experience inspired me to incorporate these insights into my thesis.

My current thesis, guided by Dr. Qing Wang, aims to address the heterogeneity issues in federated learning. I am deeply grateful for Dr. Wang's direction in the research, who, with his extensive experience, has often provided fresh perspectives when facing research challenges. I also appreciate MingKun Yang's technical support and our collaborative discussions on various solutions and research methods in related fields, which have broadened my problem-solving approach.

I am fortunate that the project has progressed beyond expectations and is moving towards publication. I am also thankful to TU Delft for providing a platform that has enabled me to learn and gain research experience, fostering my ability to practically engage with topics of interest.

Acknowledgments to Dr. Qing Wang for his invaluable guidance, MingKun Yang and Ran Zhu for their technical support, and my friends and family for their unwavering encouragement and support throughout this journey.

Tianyi Liu

Delft, The Netherlands
27th November 2023

Contents

Chapter 1

Introduction

1.1 Introduction

In the era of rapid technological advancement, Artificial Intelligence (AI) technologies have not only matured but have also been integrated into various applications, reshaping numerous industries. This growth has been significantly bolstered by the development and proliferation of intelligent devices, such as smartphones, smartwatches, and smart gateways. These devices, with their enhanced capabilities, have contributed to an exponential increase in data, which is crucial for AI-driven solutions [40]. Traditional approaches to leveraging this data have relied heavily on AI and big data techniques, involving the transfer of data from local devices to centralized cloud servers or data centers. While effective, this centralization poses significant risks in terms of data breaches and privacy invasions, as it often requires sensitive personal data to be exposed to external networks and servers.

Earlier solutions to mitigate the intense computational and storage demands on servers included distributed data parallelism. This approach entails dividing the entire training dataset into smaller, equally sized shards for distributed processing [19]. However, this approach, while addressing computational challenges, does little to alleviate concerns about data security and privacy.

With many modern devices now equipped with substantial computational capabilities, an alternative approach to machine learning has become feasible. Federated Learning (FL) emerged as a solution to these privacy concerns. Unlike traditional approaches, FL involves training local models on individual devices and then aggregating these models on a central server to form a global model. This process ensures that personal data remains on the user's device, thereby enhancing data security.

Introduced by McMahan et al. in 2016, Federated Learning began with the Federated Averaging (FedAvg) algorithm [29]. This decentralized approach enables collaborative training of a machine learning model across various devices while keeping user data localized. In this framework, each device trains a model based on its data and sends only the model parameters, not the data itself, to a central server. The server aggregates these parameters from multiple devices to form a global model. The aggregation weights are determined by the data volume on each device, which ensures that devices with more data have a greater

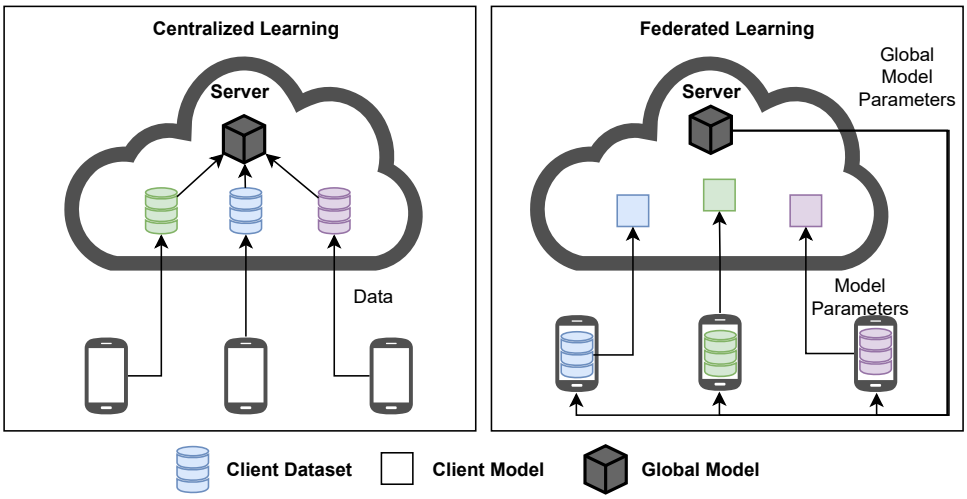


Figure 1.1: **Differences between Centralized Machine Learning and Federated Learning: Centralized Learning involves uploading clients' data to a central server, demanding significant bandwidth and posing risks to data privacy. In contrast, Federated Learning requires only the upload of locally trained model parameters, reducing bandwidth usage and better preserving clients' data privacy.**

influence on the final model.

This approach is particularly advantageous as it reduces the communication overhead compared to transmitting entire datasets and encourages users with limited local data to participate in improving the overall model performance.

Despite these advantages, Federated Learning faces challenges in practical applications, especially when dealing with non-independent and identically distributed (non-IID) data. In such scenarios, where client data distributions are unbalanced and heterogeneous, traditional FL approaches often perform suboptimally. The non-IID nature of data means that clients training solely on their datasets can achieve higher accuracy than those participating in FL, suggesting an inability of the aggregated global model to adapt effectively to individual data distributions. This limitation significantly hinders the personalization of solutions to meet the diverse needs of different clients [33].

1.2 Research Questions

Federated learning faces performance degradation when dealing with highly heterogeneous non-IID data distributions among clients. The primary issue this thesis aims to address is the global model's difficulty in adapting to local data distributions. Consequently, this thesis will propose the main research question and subsequent secondary research questions.

Main research question:

How to solve the performance degradation caused by data heterogeneity in federated learning?

Secondary research questions:

1. What causes data heterogeneity problems in federated learning?

2. What are the solutions to the data heterogeneity problem in federated learning?

3. What are the limitations of these solutions?

4. What approaches can be used to design algorithms that solve the problem of data heterogeneity?

1.3 Contribution

This thesis proposes a new training architecture, Federated Learning with Rebalanced Dataset (FedReb), based on model decoupling in personalized federated learning. It aims to achieve a well-performing global model under user data heterogeneity while ensuring no loss in performance of local models for each user, with even slight improvements.

The contributions of this thesis can be summarized as follows:

1. A training framework Federated Learning with Rebalanced Dataset(FedReb), for personalized federated learning based on model decoupling, is designed. This approach creates a new rebalanced dataset from the user's original local data without requiring additional data sample exchanges with the server. This allows the server to generate a well-performing global model while maintaining local accuracy for users.

2. A modification to the aggregation weights during the aggregation process is proposed based on the FedReb, further enhancing the global model accuracy.

3. The impact of the size and creation method of the rebalanced dataset required by FedReb during training are investigated and found an optimal solution that achieves the highest global model accuracy across various configurations.

1.4 Organization of the Thesis

To address the problems arising from client data heterogeneity and to meet individual client needs, a research direction known as personalized federated learning (PFL) has been proposed. It addresses the limitations of FL by personalizing the global model or learning personalized models locally. The goal shifts from obtaining a single optimal global model to achieving optimal local models for all clients.

One limitation of this approach is the lack of a well-performing global model, which is often a intermediate product of the training process [15]. A well-performing global model, which learns the diverse characteristics of all participating clients, would have better generalizability.

Chapter 2 will explain federated learning and the impact of data heterogeneity on it, answering secondary research question 1.

This is followed by an explanation of personalized federated learning, an effective direction for addressing data heterogeneity in federated learning, answering secondary research question 2.

As mentioned above, the limitation of PFL in lacking a well-performing global model is the answer to secondary research question 3.

Finally, to answer secondary research question 4 and ultimately resolve the main research question, Chapter 3 will propose the Federated Learning with Rebalanced Dataset (FedReb) algorithm, based on personalized federated learning. Chapter 4 will explain its implementation, and Chapter 5 will present experimental results comparing FedReb with other algorithms. In the end, Chapter 6 will provide a validation of deploying the algorithm onto the testbed.

Chapter 2

Related Work

This chapter introduces concepts related to this thesis, including federated learning, personalized federated learning, data augmentation, and MQTT. Figure 2.1 provides an overview of the differences between centralized machine learning, which combines big data with machine learning, and both federated learning and personalized federated learning. This taxonomy was proposed in the paper [33].

Centralized machine learning involves collecting user data and training on centralized servers. By learning from diverse client data, the model gains generality. However, this approach faces issues of data privacy and high communication costs due to the need to transfer user data. Additionally, when dealing with heterogeneous data, or data heterogeneity, model training can struggle to converge. Moreover, as it learns a single model, it lacks personalized solutions for local users with different data distributions and characteristics.

Federated learning allows users to train models locally and then upload these trained models to a centralized server. The server aggregates these models and redistributes them to clients. This method addresses the data privacy and communication cost issues found in centralized machine learning. However, it also faces challenges in converging when dealing with data heterogeneity and lacks personalized solutions due to its reliance on a single model.

Personalized federated learning, through methods of personalizing the global model and learning personalized models, addresses the convergence issues in federated learning caused by data heterogeneity. It also provides personalized solutions for participating clients [33].

Personalized federated learning is a research direction based on federated learning, and this thesis will explain various solutions in federated learning and personalized federated learning. The algorithm proposed in this thesis is based on the Architecture-based approaches, parameter decoupling, one of the approaches in learning personalized models strategy described in section 2.3.

2.1 Federated Learning

Federated Learning, first conceptualized by the authors of FedAvg [29], aims to train a global model that performs well across multiple clients, or an "average client." This contrasts with traditional approaches that rely on transferring local

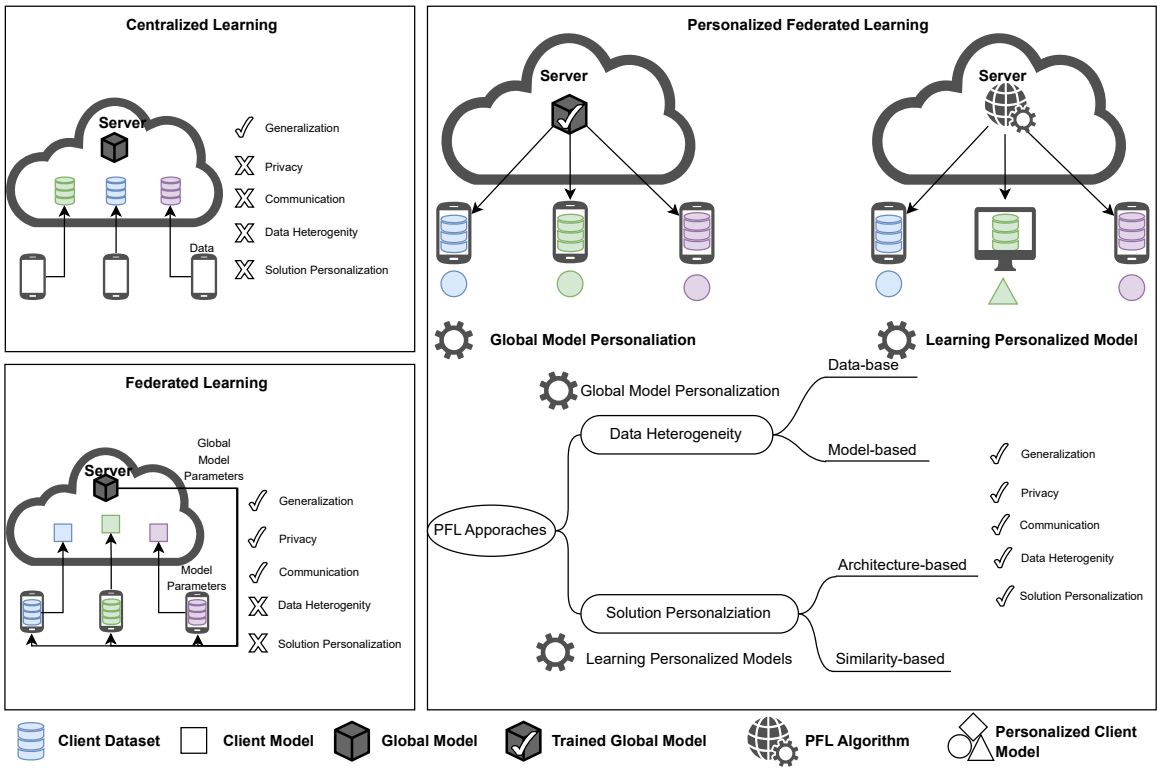


Figure 2.1: **Centrralized Learning, Federated Learning and Personalized Federated Learning.**

user data to a centralized server for training.

A typical Federated Learning training process is illustrated in Figure 2.2. During a communication round in Federated Learning, the process is as follows:

1. At the beginning of a communication round, the server sends a global model to all clients selected for that round.
2. Selected clients train the received global model on their local data.
3. Upon completing training, these clients send the parameters of their trained models back to the centralized server.
4. The server aggregates these parameters and updates them into a new global model for use in the next round.

The goal of each round is to obtain an optimal global model w that minimizes the aggregated local loss function $f_m(w_m)$ [40]:

$$f_m(w_m) = \frac{1}{D_m} \sum_{i}^{D_m} l(x_i, y_i; w_m) \tag{2.1}$$

$$\min f(w) = \sum_{m=1}^{M\ =\ C\times K} \frac{D_m}{D} f_m(w_m) \tag{2.2}$$

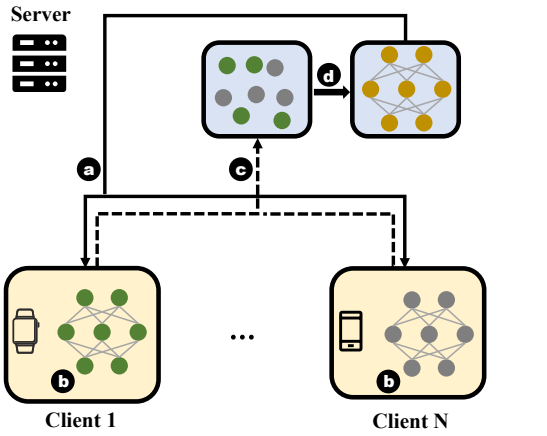


Figure 2.2: **Process of Federated Learning in one communication round.**

Here, in equation 2.1 x_i and y_i represent the features and labels of sample i , respectively. D_m is the size of the client's local dataset, D is the total size of samples from all clients participating in the round, C is the participation rate per round, K is the total number of clients,M is the client that join in this round , m is the client's ID, and l is the loss function.

Federated Learning can be broadly categorized into three types: Horizontal Federated Learning, Vertical Federated Learning, and Federated Transfer Learning. Federated Transfer Learning serves as an intermediary approach between Horizontal and Vertical Federated Learning.

2.1.1 Horizontal Federated Learning

Horizontal Federated Learning, also known as homogeneous FL, involves clients with the same feature space but different sample spaces. For instance, each client has different persons as sample data, but all the data include the same features such as Age, Sex, Height, and Weight. FedAvg [29] is an example of a Horizontal Federated Learning algorithm. In this approach, only the aggregated weights w from the server and the model weights w_m from clients participating in the round are transmitted, offering a secure way to protect data. However, as the number of training rounds increases, Federated Learning can consume substantial communication resources. Research addressing this issue includes client updates sub-sampling[13] and model quantization [35].

2.1.2 Vertical Federated Learning

Vertical Federated Learning, or heterogeneous FL, involves clients sharing the same sample space but with different feature spaces. As shown in Figure 2.4, different clients have the same samples but use different features. For example,

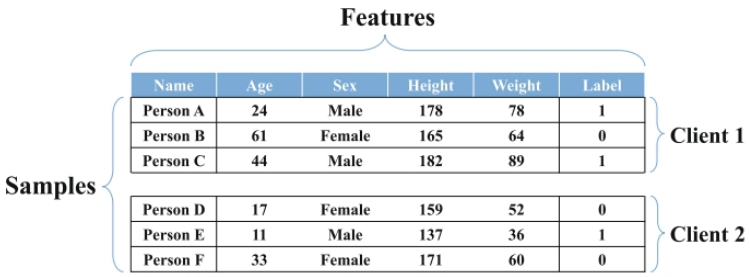


Figure 2.3: **Features pace and samples space of Horizontal Federated Learning. In HFL, every client has different data samples. These samples have the same features [40].**

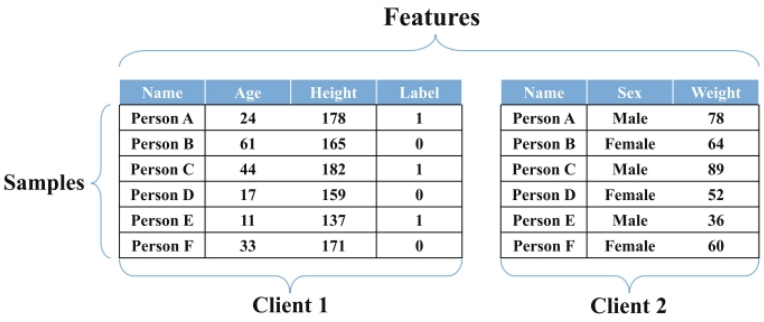


Figure 2.4: **Features pace and samples space of Vertical Federated Learning. In VFL, every client has all the data samples, but each client has different features of these data, and only the guest party has the labels of these data. Here, Client 1 is the guest party and Client 2 is the host party [40].**

Client 1's samples may have Age and Height features, while Client 2 has Sex and Weight features.

In Vertical Federated Learning, it is assumed that only one client, usually referred to as the guest party or passive party, has the data labels such as Client 1 in Figure 2.4. Other clients, known as host parties or active parties, such as Client 2 in Figure2.4, do not have these labels [40]. Unlike Horizontal Federated Learning, Vertical Federated Learning does not have a common global model. Each client uses its local model, and there is no centralized server for aggregating model parameters. However, similar to Horizontal Federated Learning, they still send some results to the guest party. The training process in Vertical Federated Learning is as follows:

1. Each selected client in a communication round trains on the same data shard with its local model, which is designed for specific features, producing a result.

2. Host parties send this result to the guest party.

3. The guest party calculates the loss function and intermediate gradients

locally, updating its model.

4. The guest party sends these intermediate gradients back to the host parties, who then update their models and repeat the process with the next data shard.

Unlike Horizontal Federated Learning, Vertical Federated Learning does not send model parameters to a central server or guest party, lacks a shared model structure and global model, and only transmits the host party's outputs to the guest party, who then sends back intermediate gradients. However, every training round and data shard training in each communication round requires communication.

Compared to Horizontal Federated Learning, Vertical Federated Learning has a performance closer to centralized machine learning and requires less data to be communicated, only transmitting client outputs and intermediate gradients rather than entire models. However, it also faces data privacy concerns [40].

Federated Transfer Learning is applied in scenarios where there is an overlap in the feature and sample spaces between Horizontal and Vertical Federated Learning.

Under the Federated Learning framework, the model performs well when the data distribution among clients is homogeneous. However, in more realistic non-IID distributions with highly heterogeneous data, this framework significantly suffers, leading to difficulty in model convergence. The accuracy of the model aggregated on the server and tested on local user data can decrease, struggling to adapt to local data distributions. This challenge of performance degradation due to data heterogeneity remains one of the significant obstacles in Federated Learning.

2.2 Client Drift : The Impact of Data Heterogeneity

The impact of data heterogeneity on Federated Learning is mainly attributed to the phenomenon of client drift, which occurs due to multiple rounds of local training on non-IID data and subsequent synchronization via a centralized server [23]. Figure 2.5 demonstrates the effect of client drift in Federated Learning algorithms on both IID and non-IID data distributions.

Taking two clients as an example, in the FedAvg algorithm, the optimal weight of the server-aggregated global model gradually shifts towards the average of the clients' optimal solutions. When the data is IID distributed, the global model's optimal solution w^* is equidistant from each client's optimal solutions w_1^* and w_2^*. In this process, the global model w^t with each round of aggregation gradually approaches the global optimal solution w^*.

However, with non-IID data distribution, the theoretical global optimal solution w^* is no longer equidistant from the clients' local optimal solutions. As shown in Figure 2.5(b), the global optimal solution w^* in this case is closer to Client 2's optimal solution w_2^*.

Despite multiple rounds of updates in the FedAvg algorithm, clients' weights tend to converge towards their local optimal solutions, but the global model weight w^t still approaches the average of the local clients' optimal solutions.

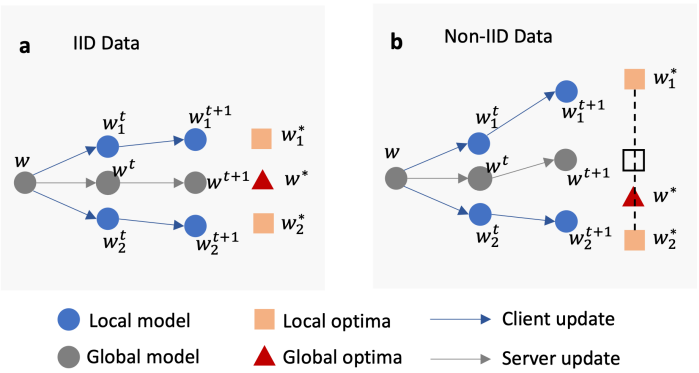


Figure 2.5: **Phenomenon of Client Drift [33].**

This result deviates from the theoretical global optimal solution, leading to convergence issues.

2.3 Personalized Federated Learning

Personalized Federated Learning is a research direction proposed on the basis of traditional federated learning, aimed at addressing the convergence challenges in highly heterogeneous data environments while meeting individual user needs. Unlike federated learning, personalized federated learning does not solely focus on developing a global model that performs well on an "average client." Instead, it emphasizes cultivating models that perform well locally, tailored to each client's data distribution [26].

This field mainly employs two approaches: personalization of the global model and learning personalized models. As illustrated in Figure 2.6, a distinction is made between generic machine learning, federated learning, and personalized federated learning. Personalized federated learning primarily involves two strategies: **Global Model Personalization** and **Learning Personalized Models**, each further divided into various approaches. [33].

2.3.1 Global Model Personalization

The first category, Global Model Personalization, follows the traditional federated learning training approach. It addresses the issue where a single global model underperforms in highly heterogeneous data contexts. Similar to traditional federated learning, this strategy initially develops a single global model.

However, post distribution to clients, the global model undergoes localized adaptation training based on individual data distributions. This dual process of federated learning training followed by local adaptation is a common strategy in personalized federated learning [22].

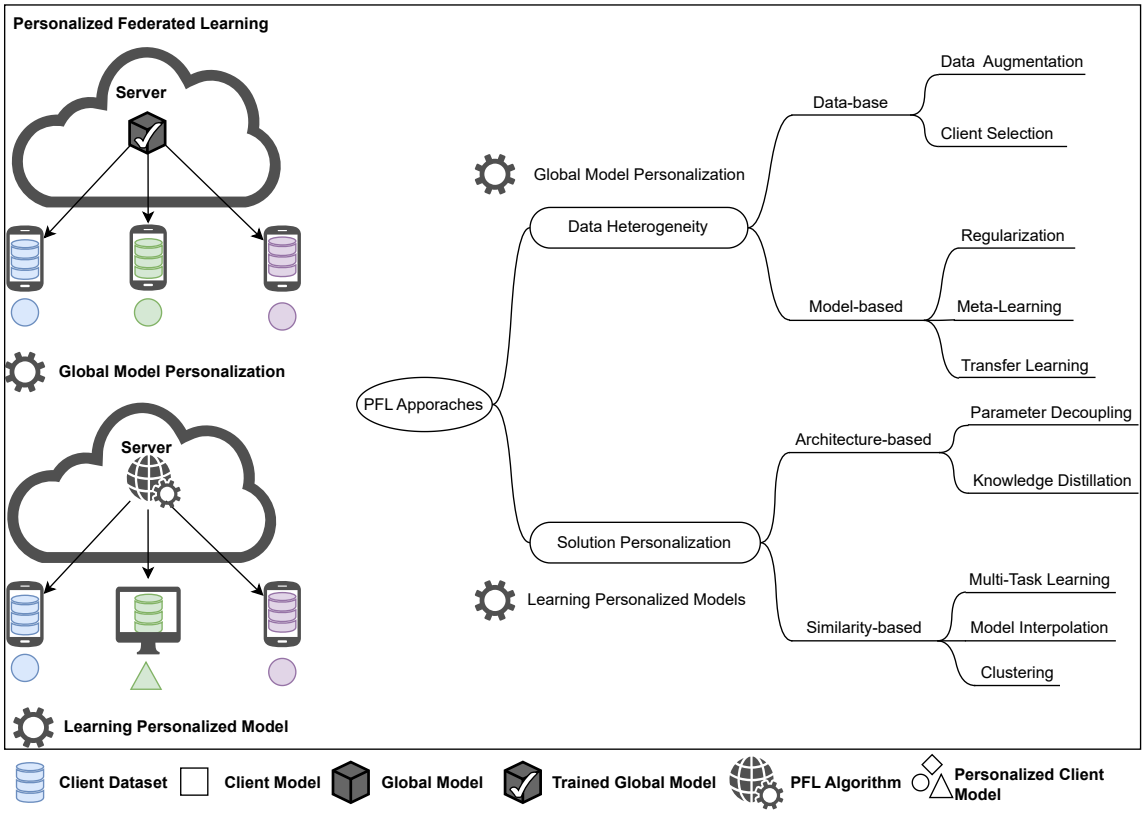


Figure 2.6: **Personalized Federated Learning Approaches. The algorithm proposed in this thesis is based on Parameter Decoupling, which is one of the Architecture-based approaches.**

The performance of the localized model under this strategy depends heavily on the global model's effectiveness. Therefore, many approaches strive to enhance the global model's performance in highly heterogeneous data environments to improve subsequent localized adaptive personalization.

As Figure 2.7 depicted, the Global Model Personalization strategy is categorized into two approaches: Data-Based and Model-Based.

Data-Based Approaches

Data-based approaches aim to reduce the statistical heterogeneity of client data. These approaches include:

1. **Data Augmentation**: Considering IID data distribution as a fundamental assumption in statistical learning theory, the goal of data augmentation is to enhance the statistical homogeneity of user data. To address unbalanced data distribution, techniques like Over Sampling [21]and Under Sampling [25] have been proposed. However, direct application in federated learning is challenging due to data privacy concerns. Therefore, data augmentation typically requires some degree of data sharing or a

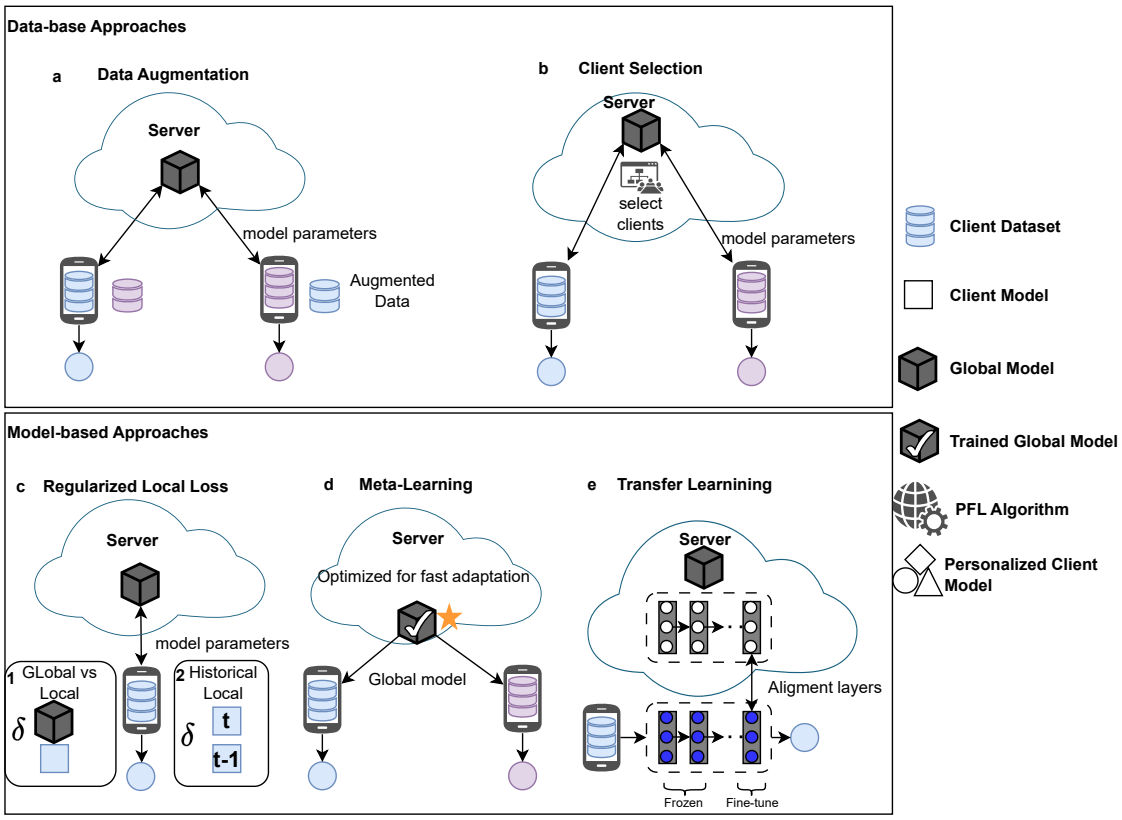


Figure 2.7: **Strategies for Implementing Personalized Federated Learning with Global Model Personalization. The figure illustrates various approaches to personalizing the global model in personalized federated learning. a) Data Augmentation b) Client Selection are data-based approaches. c) Regularized Local Loss can be achieved through 1) regularization between the global model and local models or 2) regularization between local models and their respective ideal models. d) Meta-Learning e) Transfer Learning are model-based approaches [33].**

representative dataset that mirrors the overall data distribution [33].

2. **Client Selection**: This approach implements a client selection mechanism designed to balance data distribution among selected clients, thereby enhancing the global model's performance [34].

Model-Based Approaches

Model-Based approaches focus on two objectives: acquiring a robust global model for subsequent personalization and enhancing the model's adaptability to individual local environments. Specific approaches include:

1. **Regularized Local Loss**: Regularization in loss functions is commonly used to prevent overfitting and improve convergence. In personalized fed-

erated learning, regularized local loss functions limit the deviation of the model post-local training, indirectly enhancing the stability and speed of global model convergence. Notable algorithms include FedProx [28] and SCAFFOLD [23].

2. **Meta-Learning**: Often referred to as "learning to learn," meta-learning aims to optimize algorithms through exposure to various tasks or datasets. In personalized federated learning, non-IID data distributions from different clients are treated as distinct tasks. The server aggregation process is akin to the training phase in meta-learning, while local adaptive training resembles the testing phase, applying gradient descent based on local data distribution. Renowned algorithms in this category include Per-FedAvg [20].

3. **Transfer Learning**: A common technique in machine learning, transfer learning aims to transfer knowledge from one domain to another similar one. In personalized federated learning, the global model often starts as a pre-trained model, which clients then adaptively train locally for personalization.

2.3.2 Learning Personalized Models

The central aim of this strategy is to address the personalization of the solution itself. Contrary to the strategy of global model personalization, which focuses on training a singular global model, this approach emphasizes training a separate, personalized federated learning model for each client. This often involves modifications to the federated learning framework or process to achieve personalized models or solutions. As shown in the Figure 2.8, the strategy of learning personalized models mainly divides into two categories: Architecture-Based Approaches and Similarity-Based Approaches.

Architecture-Based Approaches

Architecture-Based solutions achieve personalization by allowing each client to have its own personalized model. The main approaches include Parameter Decoupling and Knowledge Distillation:

1. **Parameter Decoupling**: This approach involves segmenting a part of the model to serve as the personalized component for each client. The personalized segment is trained locally with client-specific data and is not transmitted to the centralized server nor updated by the aggregated model. This allows the personalized portion to be tailored to local data distributions or even specific tasks. Decoupling strategies typically follow two types: "**base layers + personalized layer**," exemplified by algorithms like FedPer [12]. The personalized head layer learns specific expressions based on local data distribution or tasks, while the base part acts as a feature extractor, learning low-level generic features of the samples. Another type involves each client retaining a private personalized feature representation, while the rest of the model is transmitted to a centralized server for aggregation.

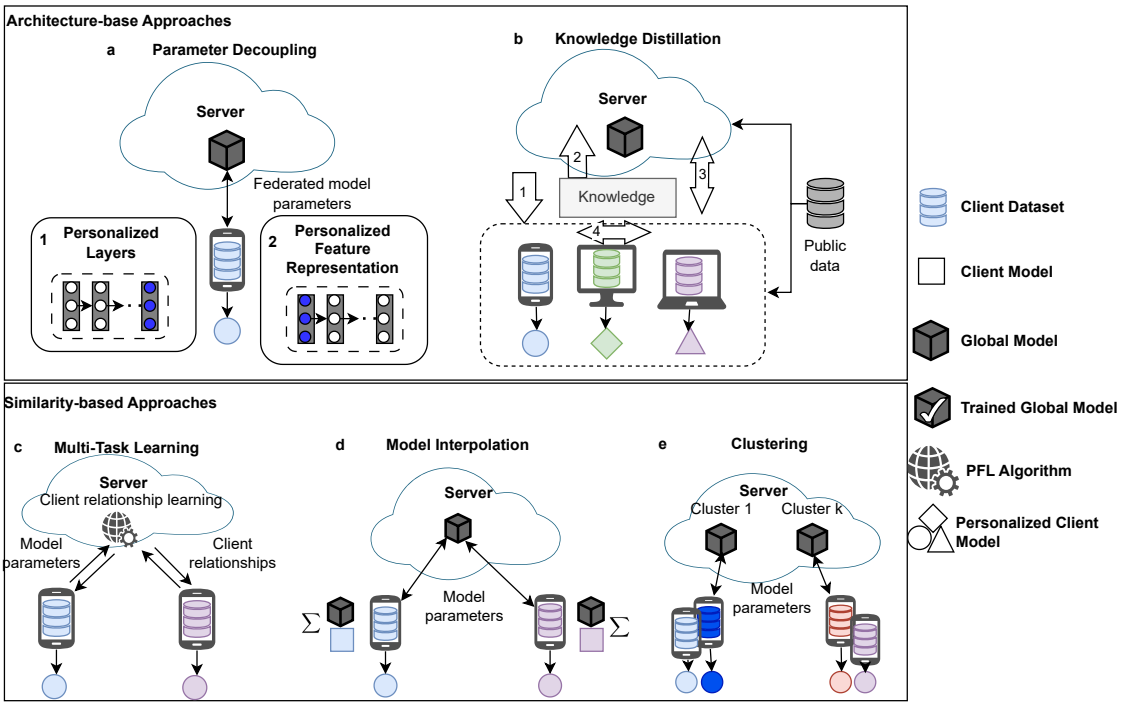


Figure 2.8: **Strategies for implementing personalized federated learning with Learning personalized models include these approaches: a) Parameter Decoupling, achieved by retaining parts of the model as private, either through 1) personalized head layers or 2) personalized feature representation, and b) Knowledge Distillation, where knowledge transfer can occur through 1) client to server distillation, 2) server to client distillation, 3) mutual distillation between client and server, and 4) mutual distillation among clients. These two are architecture-based approaches. c) Multi-task learning, d) model interpolation, and e) clustering are considered similarity-based approaches [33].**

2. **Knowledge Distillation**: Knowledge Distillation transfers knowledge from a group of teacher models to a lighter student model. The "knowledge" usually pertains to class scores or logit outputs for a sample, indicating the probabilities or scores for all possible classes. For instance, in a 10-class task, a sample processed by the model would yield ten probabilities, each corresponding to the likelihood of belonging to one of the ten classes. Knowledge distillation involves training a simpler structure to emulate the output of a more complex model. In personalized federated learning, this allows each client to have entirely different models based on their local data distribution or task type. Generally, knowledge distillation requires a common dataset for different models to imitate the results on this dataset. There are four main ways to achieve personalized federated learning through knowledge distillation [33]:

   (a) Client distills knowledge from the server to obtain a stronger person-

alized model,

(b) The server distills knowledge from clients to improve the global model,

(c) Bidirectional distillation between clients and server,

(d) Mutual distillation among clients without server involvement.

Similarity-Based Approaches

Similarity-Based approaches require establishing a relationship of similarity among clients. Clients learn from others within this relationship to develop a localized personalized model. Specific techniques include Multi-task Learning, Model Interpolation, and Clustering:

1. **Multi-task Learning**: This approach trains a model to adapt to multiple related or similar tasks, enabling the model to learn common features and key knowledge across different tasks to enhance its generality. In personalized federated learning, different clients with non-IID data distributions can be viewed as similar tasks in multi-task learning.

2. **Model Interpolation**: This technique balances the local model of the user with the aggregated global model. It considers the similarity between the local and global models, introducing a penalty parameter λ to prevent large deviations between the two. When λ equals zero, it means each client trains solely based on their local data, independent of the aggregated model. As λ increases, the local model becomes more similar to the global model. When λ approaches infinity, all local models nearly resemble the global model, eliminating personalization and resembling traditional federated learning.

3. **Clustering**: Clustering is an approach used when there is significant heterogeneity and variance in client data. It groups clients with similar data into clusters, employing the same model within each cluster and different models across clusters. This strategy assumes a natural grouping or clustering of clients based on data distribution.

The approach to personalized federated learning is diverse, and various approaches can be combined to create new solutions. However, in traditional federated learning algorithms, the global model often serves merely as an intermediate product during training [15]. This overlooks the potential of federated learning to develop a versatile global model that learns from the diverse local data of each client.

The approach adopted in this thesis is based on the data decoupling approach in learning personalized model strategy, while also incorporating a data augmentation solution. Data decoupling is achieved by splitting the training model into parts for aggregation on the server and parts for training and use on the client's local device. The following chapter will provide detailed descriptions of the approaches that have been designed.

2.4 Data Augmentation

Data augmentation, while not the central focus of this thesis, plays a significant supportive role in enhancing the federated learning model's effectiveness.

In deep learning, especially in computer vision and image processing, data augmentation is crucial. It involves expanding the dataset artificially through modifications to the original data, such as geometric transformations and noise addition [18]. Data augmentation's primary purpose is to address data scarcity, a

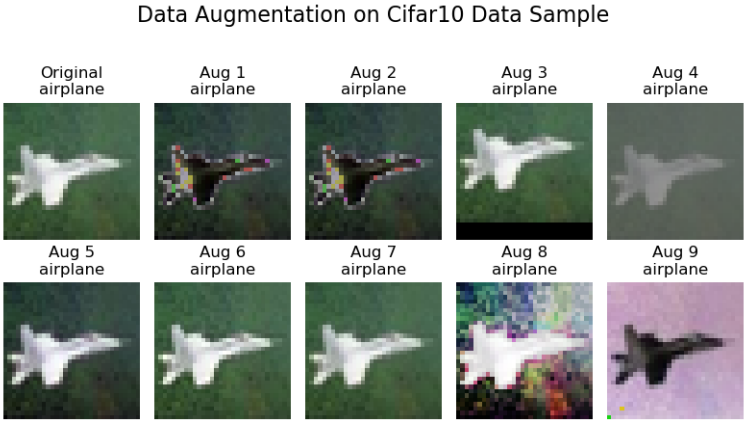


Figure 2.9: **Data Augmentation on Cifar10 Data Sample.**

common challenge in deep learning. Gathering extensive datasets can be costly and time-consuming. Data augmentation offers a low-cost, efficient solution to reduce reliance on extensive data collection and preparation [32]

In federated learning, data augmentation mitigates data heterogeneity, leading to more evenly distributed and similar data distributions across clients while maintaining privacy [18]. Augmented data significantly improves the performance and results of deep learning models [32].

Data augmentation also reduces operational costs related to data collection and labeling [32]. In the thesis, Data augmentation is used to generalize the rebalanced dataset based on local datasets and improve the generalization of the global model when aggregated.

2.5 Message Queuing Telemetry Transport

Message Queuing Telemetry Transport (MQTT) is a messaging protocol based on a publish-subscribe mechanism [30]. In the thesis, it is primarily used to establish a TestBed for communication between simulated servers and clients on experimental devices. In the MQTT protocol, there are two main network entities: the Message Broker and the Clients. The role of the Message Broker is to receive messages published by clients and forward them. After connecting to the broker, each client subscribes to its chosen topics. A client can subscribe to multiple topics, and messages sent to a specific topic are forwarded by the message broker to all other clients subscribed to the same topic. MQTT was selected for this thesis mainly due to its mechanisms like Persistent Session, Retained Messages, and Last Will and Testament (LWT). These features effectively handle issues such as device crashes and disconnections during experiments, allowing

the training to continue after reconnection.

2.5.1 Persistent Session

This mechanism allows clients to set their Quality of Service (QoS) level, with three levels available: 0, 1, and 2. At QoS levels 1 or 2, the message broker retains the client's subscribed topics and unacknowledged messages even after disconnection. When the client reconnects, these messages are resent. At QoS level 0, however, if the client disconnects, new messages published on subscribed topics during the disconnection are not retained, and the client won't receive these messages upon reconnection. This is particularly useful for handling poor network conditions, preventing data transmission interruptions, and ensuring data completeness for ongoing processes.

2.5.2 Retained Message

With this feature, the latest message marked as "Retained" under a topic is preserved. When a new client subscribes to this topic, the retained message is immediately sent to them. This ensures that when clients connect to the server, they receive the experiment details immediately without requiring the server to resend these details repeatedly.

2.5.3 Last Will and Testament (LWT)

This mechanism sends a LWT message to notify other clients when a client disconnects unexpectedly. It is crucial for addressing situations where an experimental device crashes or disconnects, preventing the server from unknowingly selecting the disconnected client for further training, which could otherwise lead to process interruptions.

Chapter 3

FedReb Overview

This chapter will provide a detailed explanation of the design philosophy and specific processes of the newly designed framework. It will first introduce the fundamental ideas and key assumptions underlying the algorithm design, as well as how the non-IID data used in the simulation is partitioned. This will be followed by a general description of the algorithm, illustrated with pseudocode to explain the overall process.

3.1 Basic Idea / Key Assumptions

3.1.1 Feature Extractor

This part mainly explains the feature extraction characteristics of different parts of the model for tasks or data. As mentioned earlier, FedPer proposes a framework for personalized federated learning through parameter decoupling, consisting of "**base layers + personalized head layers**" [12]. In this framework, the base layers are aggregated through federated learning, while the personalized head layers are retained by the clients and trained with local data.

This framework assumes that the base layers of the model can learn low-level, general-purpose features that are common across different user data distributions or similar tasks, such as color blocks, textures, corners, or edges in images. These features are common regardless of how different the data distributions are. Therefore, it is feasible to design a structure where the base layers participate in federated learning to learn features from various data distributions.

In contrast, the remaining layers, referred to as personalized head layers in FedPer, are opposite to the base layers [12]. They learn information useful for specific tasks and data distributions, based on different combinations of low-level features extracted by the base layers, such as specific shapes formed by combinations of textures, colors, and edges in image information. These layers learn classifications specific to data distributions or tasks.

The understanding and assumptions about base and personalized head layers form the foundation of this thesis. The thesis also adopts the parameter decoupling approach of splitting the model into base and personalized head layers.

However, unlike FedPer [12], where only the base layers participate in federated learning, the new approach introduced in this thesis additionally involves a

head layer, inspired by FedROD [15], allowing the server to use it to aggregate a complete model for task completion. This additional head layer is trained locally with the client's original personalized head layer but uses a different dataset, which will be explained in detail later.

Besides FedPer, other algorithms like FedRep [16], FedBABU [31], and FedROD [15] also achieve personalized federated learning through parameter decoupling. These algorithms split the model into two parts: one part is completely private and trained locally by the client, and the other part is aggregated and updated through federated learning.

The FedROD [15]algorithm mentioned above is another algorithm considered as the reference in this thesis. It introduces an additional head for global aggregation and a softmax balanced loss, achieving a well-performing global model while addressing data heterogeneity and fulfilling diverse personalized solutions for different clients.

3.1.2 Data Distribution

In federated learning, the assumed independently and identically distributed (IID) data implies that the data volume is similar among all participating clients, and the distribution of various types of data is also similar. Specifically, if a client has 100 samples with 10 classes, each class having 10 samples in a uniform distribution, other clients would also have approximately 100 samples with a similar distribution. This is an ideal assumption and hard to achieve in reality.

When data distribution is non-IID, closer to real-world scenarios, the performance of federated learning is not as satisfactory. In personalized federated learning, to simulate the non-IID situation in real-world data distributions, various methods are used to allocate data to simulated users.

One common situation in personalized federated learning that needs to be addressed in non-IID is label distribution skew. Suppose a sample data (x, y), where x represents features like pixel position and color in an image input to the model, and y is the label of the image. The data distribution for a local user m is $P_m(x, y)$.

In cases where label distributions differ, each client's label distribution $P_m(y)$ is different. However, the feature distribution for the same label should be similar, meaning that although local data distributions differ, the features are the same, i.e., the conditional distribution $P_m(x \mid y)$ is similar [40].

Label distribution skew mainly involves differences in the number of samples per client and the distribution of samples across classes. In the original FedAvg paper, a method to simulate unbalanced data distribution is proposed, where each client only has data for specific c classes[29].

Figure 3.1 shows the situation where each client only has data for two classes. When c is set equal to the total number of classes in the dataset, the data distribution among clients approximates an IID distribution. This approach is also known as the "hold c classes" scheme.

Another approach that closely simulates the non-IID distribution among clients is based on the Dirichlet distribution $Dir(\alpha)$. It assumes that the probability of a class c being distributed to client m is $p_{c,m}$.

For a client, the probability of class c being distributed is $p_c \sim Dir_k(\alpha)$. The smaller the α, the more imbalanced the client data distribution; as α approaches infinity, the data distribution among clients becomes IID.

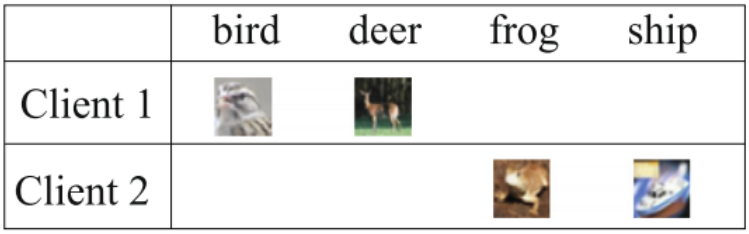


Figure 3.1: **Data partition method: Hold c classes [40]**

The bar chart below Figure 3.2 shows the distribution of a 10-class dataset among 20 clients according to the Dirichlet distribution with $\alpha = 0.1$, where each bar represents a client, each color represents a class, and the length of the bar indicates the quantity of data. The method used in this thesis to simulate non-IID data distribution also adopts this approach.

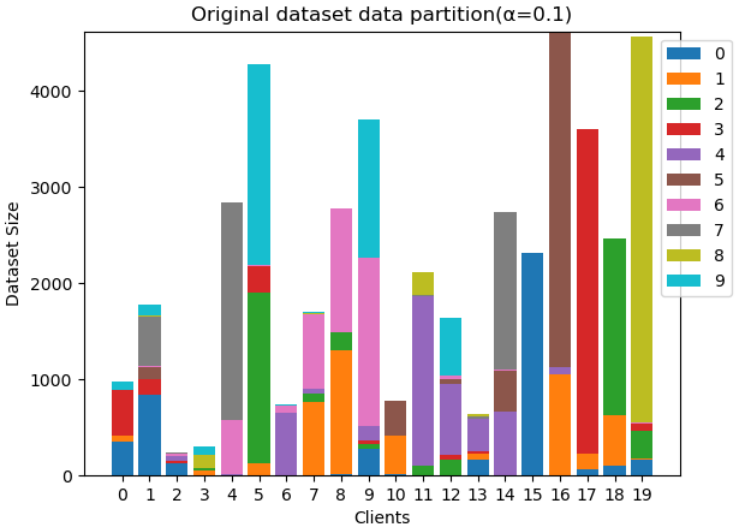


Figure 3.2: **Data partition method: Dirichlet distribution**

3.2 Overview of the FedReb Algorithm

This section divides the algorithm into two parts for explanation: first, the aggregation part in federated learning, and then the training process for individual clients. However, before that, an explanation of how to generate the Rebalanced Dataset mentioned in the algorithm is provided.

3.2.1 Rebalanced Dataset

The Rebalanced Dataset is generated based on the local original dataset. It first determines a value based on the number of each class of data in the original dataset. This value can be the mean number of data for classes that have data in all users' local original datasets. For example, if a user only has data in 5 classes, then this mean value would be the mean of these 5 classes. This value could also be the maximum, median, or minimum value. The main purpose is to choose a threshold value t, so that the data of all held classes increases or decreases to t, achieving a balance among the existing classes, approaching the effect of an IID dataset. However, it might become a scheme similar to the "hold c class" mentioned earlier due to the limited number of classes in local data, but since the classes held by each client vary, it is not exactly the same as that scheme.

Once this value is determined, classes exceeding this value will randomly select t samples, while those below t will use data augmentation to increase their samples to t. Compared to the original local dataset, the Rebalanced Dataset has the same classes as the original local dataset, but the quantity of data in each class is the same, all set to a specific threshold value t.

The main purpose is to train a more balanced model locally and make the model aggregated through federated learning more balanced and generalizable. Figure 3.3 shows the clients' data distributions of the rebalanced dataset generated with the threshold t. The threshold t is set to the mean number of samples across classes that have non-zero samples in the original dataset. The original dataset is generated based on the Dirichlet partition in Figure 3.2.

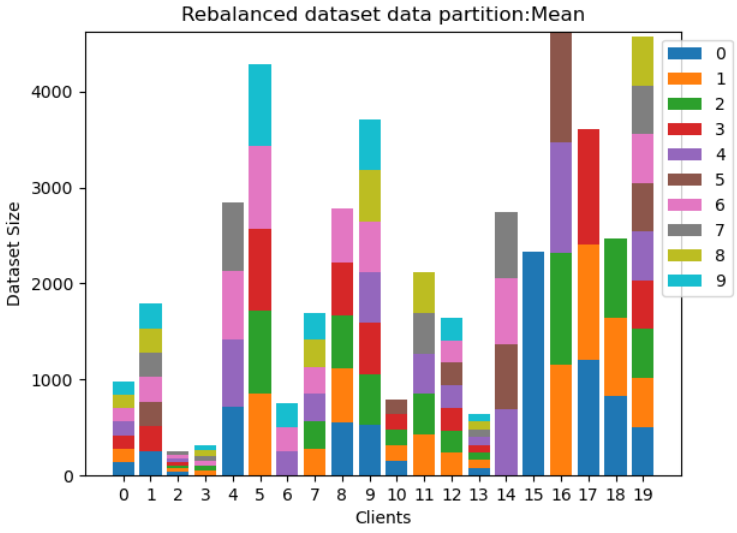


Figure 3.3: **Rebalanced Dataset Generated with threshold t as mean value.**

3.2.2 FedReb Client: Local Training Part

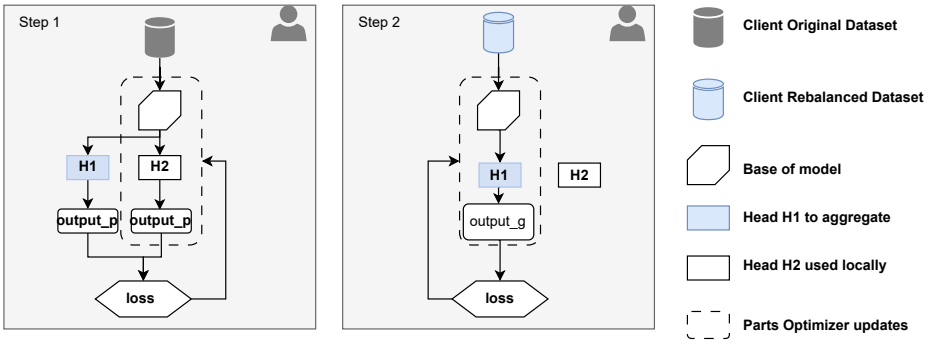


Figure 3.4: **Overview of Client Local Training Part of FedReb**

As shown in Figure 3.4, the local training process of the user consists of two steps, with the model being trained sequentially through two sets of data. Training starts with the original dataset, where data is processed through the base layer to output an intermediate result, *mid*. *mid* is then processed through Head $H1$ and Head $H2$, respectively outputting $output_g$ as a prediction result with a more balanced distribution among classes and $output_p$ as a prediction result where the distribution among classes depends on the local data distribution.

"$output_p$+$output_g$" is used as the output result for input into the loss function to calculate the gradient. During backpropagation, the gradient is propagated back to the base through both $H1$ and $H2$. Subsequently, the optimizer updates the parameters of the Base and Head $H2$, while $H1$ remains unchanged. The idea of using "$output_p + output_g$" as the output result is to allow Head $H2$, which is closer to the local data distribution, to be trained more quickly with the help of Head $H1$, which is aggregated through federated learning and has a more balanced and generalizable prediction for various classes.

Next, the Rebalanced Dataset is used to retrain the base. The output mid result is only fed to $H1$, and the loss is calculated only based on $H1$'s $output_g$. During backpropagation, it only propagates through $H1$ to the base. The optimizer then updates the base and $H1$.

After this, the base and $H1$, as a complete model, upload their parameters to the server for aggregation. This step mainly aims to train a Head $H1$ with a more balanced prediction result distribution through a Rebalanced Dataset with a more balanced data distribution, providing the central server with a model that has strong generalizability.

This step updates the Base part because the base is considered a low-level feature extractor, and the low-level features learned from the Rebalanced Dataset generated based on the original dataset are not significantly different from those learned from the original dataset.

Moreover, the Rebalanced Dataset includes samples generated through data augmentation, which should make the base part more robust.

For this training approach, it is worth noting that the number of training samples for the local model's base part and Head $H1$ are actually different because the rebalanced dataset could be created with different threshold t. There-

fore, some adjustments to the server aggregation step mentioned earlier are proposed in this thesis.

3.2.3 FedReb Server: Federated Learning Part

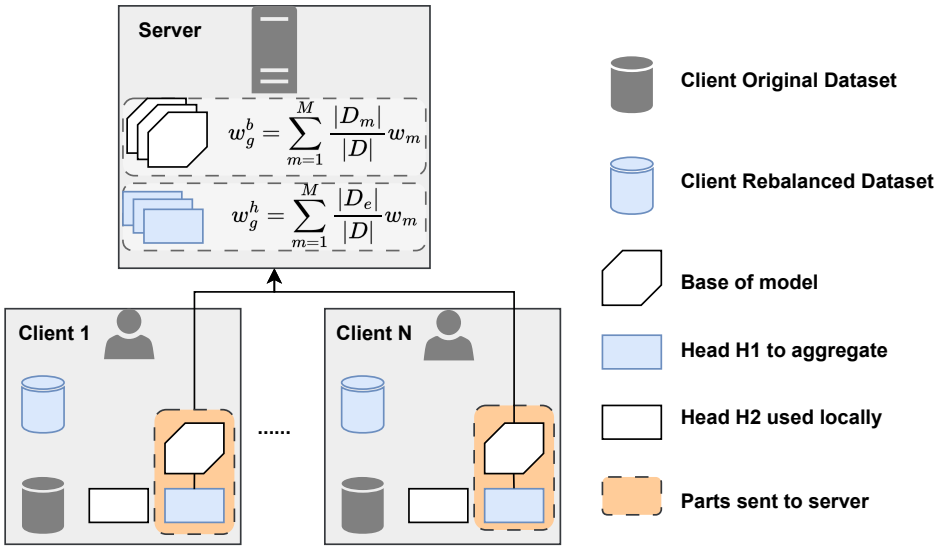


Figure 3.5: **Overview of Federated Learning Part of FedReb**

As shown in Figure 3.5, an explanation of all components in the entire process is provided. Each client holds their own original dataset and a Rebalanced Dataset generated from it. The model structure held by each client is the same, with each having a base layer of the model, and Head layers $H1$ and $H2$. By combining the base layer with the Head layers, the model can make complete predictions and learn about each category of a task or dataset, meaning the Head layers will output for every class of the dataset, regardless of whether the local user has data in those classes.

The Head layer $H1$ is trained with the local Rebalanced Dataset, resulting in $H1$ being influenced by a dataset that is closer to an IID distribution, leading to more balanced model predictions. When a certain class significantly exceeds others in quantity, machine learning models tend to predict future outcomes as the class with more data in the training set.

However, by training Head $H1$ with the Rebalanced Dataset, the predictions made by combining the base layer and Head $H1$ will not be biased towards a specific class due to unbalanced local data distribution. Subsequently, Head $H1$, along with the base layer, is transmitted to the centralized server for aggregation, resulting in a model with strong generalizability.

The Head layer $H2$ is trained with the local non-IID distributed original data, learning the characteristics of the local data distribution. Thus, predictions made by combining the base layer and Head $H2$ will be more aligned with the local data distribution. After training, Head $H2$ is retained locally and not updated through federated learning. When predicting locally, both Head $H1$ and $H2$ work together.

During server aggregation, the aggregation weights differ from traditional federated learning. They are based on the size of the original datasets of the clients participating in the current round. Typically, the server's aggregation part adopts the FedAvg aggregation approach, as written in the following equation 3.1:

$$w_g = \sum_{m=1}^{M} \frac{D_m}{D} w_m \tag{3.1}$$

In equation 3.1, w_g is the global model's parameters, w_m is the parameters of client m's model, D_m is the size of client m's original dataset, D is the total size of the original datasets of all clients participating in the current round, and M is the number of clients participating in the round. However, in this thesis, the model's base part and Head part are aggregated separately with different weights, as shown in the following formulas:

$$w_{b,g} = \sum_{m=1}^{M} \frac{|D_m^o|}{|D^o|} w_{b,m} \tag{3.2}$$

$$w_{h,g} = \sum_{m=1}^{M} \frac{|D_m^e|}{|D^e|} w_{h,m} \tag{3.3}$$

Here, in equations 3.2 and 3.3,$w_{b,m}$ is the weight of each user's base,$w_{b,g}$ is the weight of the aggregated global model's base, $w_{h,m}$ is the weight of each client's Head $H1$, and $w_{h,g}$ is the weight of the aggregated global model's head. D_m^o is the size of client m's original dataset, which is the previously mentioned D_m. D^o is the same as D, the total size of the original datasets of all clients participating in the current round.

D_m^e refers to the number of effective samples in client m's Rebalanced Dataset, and D^e refers to the total number of effective samples in the Rebalanced Datasets of all clients participating in the current round.

Effective samples are actually the number of samples obtained in the Rebalanced Dataset that are not generated through data augmentation. If the target value t is the mean, then the number of effective samples is calculated as follows equation 3.4:

$$D_m^e = \sum_{i}^{k} e_i \tag{3.4}$$

In equation 3.4, k is the total number of classes, i is the class label, and e_i is the number of effective samples in class i in the Rebalanced Dataset. For classes in the original dataset with fewer samples than t, e_i is the number of samples in class i, as the other data are generated through data augmentation. For classes with equal to or more than t, e_i is t, as the data for these classes are reduced to t.

Algorithm 1: Algorithm of Rebalanced Dataset

```
Input: Client Local Dataset
for Class label i = 0, 1, 2, ··· , k do
    e_i ← Count quantity of data of class i
end
for Class label i = 0, 1, 2, ··· , k do
    if 0 < e_i < t then
        Generate t − e_i augmented data base on existed class i data
        Add augmented data to the Rebalanced Dataset
        Add original data fo class i to the Rebalanced Datset
    else
        if e_i ≥ t then
            Random Select t data from class i
            Add selected data to the Rebalanced Dataset
        end
    end
end
```

3.3 Alogrithm of Federated Learning with Rebalanced Dataset

3.3.1 Algorithm of Creating Rebalanced Dataset

The algorithm for generating rebalanced datasets begins by reading the user's original local data d_o and initializing an empty dataset or container d_r to receive samples for the rebalanced dataset. Next, the quantity of data for each class is calculated to compute the threshold t. Here, t is set as the mean value of the number of samples of all non-zero classes.

For the i class in the original dataset, if its sample size e_i is between 0 and the threshold t, data augmentation is used to generate "$t - e_i$" samples to add to the rebalanced dataset d_r, followed by adding the original data of class i to d_r. This results in class i in d_r having a total of t samples.

For classes in the original dataset with a sample size e_i greater than or equal to t, t samples are randomly selected and added to the rebalanced dataset. As the threshold t is the mean value of all classes with non-zero samples, the size of the rebalanced dataset is the same as the original dataset.

3.3.2 Algorithm of FedReb Client

The client-side local training algorithm begins by initializing the loss function l, optimizers opt_1 and opt_2, and the local model w_m, including the base model parameters $w_{b,m}$ and the parameters of two heads: head $H1$ for aggregation $w_{H1,m}$ and the personalized head layer head $H2$ $w_{H2,m}$. It loads the original dataset $d_o$and the rebalanced dataset d_r.

At the start of each communication round k, if the client is selected, it receives the global model $w_g^{(k)}$ from the centralized federated learning server. The global model's base parameters$w_{b,g}^{(k)}$ and head parameters $w_{h,g}^{(k)}$are loaded into the local

Algorithm 2: Algorithm of FedReb Client

Input: loss function l, optimizer, original data loader d_o, rebalance data loaderd_r

Initialzied w_m at random
for *communication round k = 1,2,3,...* **do**
  if *Client m is selected* **then**
    Receive $w_g^{(k)}$ from Server
    for *batch* $b_o = (x_o, y_o)$ *in* d_o **do**
      mid $\leftarrow$ base(x_o)
      $output_{global} \leftarrow$ head $H1$(mid)
      $output_{persoanl} \leftarrow$ head $H2$(mid)
      loss $\leftarrow$ f($output_{global}$+$output_{persoanl}$,y_o)
      loss function backwards
      Optimizer updates $w_{b,m}^{(k)}$,$w_{H1,m}^{(k)}$,$w_{H2,m}^{(k)}$
    end
    for *batch* $b_r = (x_r, y_r)$*in* d_r **do**
      mid $\leftarrow$ base(x_e)
      $output_{global} \leftarrow head_1$(mid)
      loss $\leftarrow$ f($output_{global}$,y_r)
      loss function backwards
      Optimizer updates $w_{b,m}^{(k)}$,$w_{H1,m}^{(k)}$
    end
    Send D_m^o,D_m^r
    Send $w_{b,m}^{(k)}$, $w_{H1,m}^{(k)}$ to Server
  end
end

model's base $w_{b,m}$ and head $H1$, $w_{H1,m}$, replacing the local base and head $H1$ parameters.

Training starts with loading data from the local original dataset d_o, loading a batch b_o of data x_o and their corresponding labels y_o. The data first passes through the local model's base to generate an intermediate result mid, which then goes through heads $H1$ and $H2$ to output predictions $output_g$ and $output_p$, respectively.

"$output_g$ + $output_p$" is used as the model's output, which, along with the sample labelsy_o, is input into the loss function l to calculate gradients and perform backpropagation through the entire model. Gradients for the model's base, head $H1$, and $H2$ are calculated.

Then, optimizer opt_1 updates the model's base and head $H2$ based on the gradients from the loss function, while head $H1$ is not updated.

Next, data from the rebalanced dataset d_r is loaded for training, loading a batch b_r of data x_r and corresponding labelsy_r. This data also passes through the local model's base to generate an intermediate result mid, which then goes through head $H1$ to produce $output_g$.

$output_g$ and the data labels y_r are input into the loss function l to calculate loss and gradients, and backpropagation is performed on the model's base and

head $H1$. Finally, optimizer opt_2 updates the model's base and head $H1$.

After training, the client sends the model's base and head $H1$ as a complete model to the centralized server.

3.3.3 Algorithm of FedReb Server

Algorithm 3: Algorithm of FedReb Server

Input: Initialzied w_g at random

for *Communication round k = 1,2,3,...* **do**

- Select Client join in this round M
- Send $w_g^{(k)}$ to Client $m \in M$
- Receive D_m^o,D_m^r
- Receive $w_{b,m}^{(k)}$, $w_{H1,m}^{(k)}$ from Client $m \in M$
- Calculate Clients' aggregation Weight $\gamma_{b,m} = \frac{D_m^o}{D^o}$, $\gamma_{h,m} = \frac{D_m^r}{D^r}$
- Aggregate global model base with $w_{b,g}^{(k+1)} = \sum_{m=1}^{M} \gamma_{b,m} \times w_{b,m}^{(k)}$
- Aggregate global model head with $w_{h,g}^{(k+1)} = \sum_{m=1}^{M} \gamma_{h,m} \times w_{H1,m}^{(k)}$

end

From the perspective of a centralized server, the system's algorithm begins with the server initializing a complete model parameter w_g. Then, in each communication round, the server randomly selects multiple client IDs to form the set M and sends the aggregated model parameters $w_g^{(k)}$ from the previous round to the selected clients $m \in M$. The model parameters sent in the first round are the randomly initialized w_g.

After sending the model parameters$w_g^{(k)}$ for communication round k, the server waits for all selected clients to complete local training. Clients $m \in M$ send back the size of their original local training dataset D_m^o, the size of the effective samples in the rebalanced dataset D_m^e, the base part of the local model $w_{b,m}^{(k)}$, and the parameters of head $H1$ for aggregation $w_{H1,m}^{(k)}$ to the centralized server. The server calculates the weights $\gamma_{b,m}$ and $\gamma_{h,m}$ for aggregating the global model's base and head, with equations 3.7and 3.8respectively:

$$D^o = \sum_{m \in M} D_m^o \tag{3.5}$$

$$D^e = \sum_{m \in M} D_m^e \tag{3.6}$$

$$\gamma_{b,m} = \frac{D_m^o}{D^o} \tag{3.7}$$

$$\gamma_{h,m} = \frac{D_m^e}{D^e} \tag{3.8}$$

The server aggregates the local model's base parameters $w_{b,m}^{(k)}$ and head $H1$'s parameters $w_{H1,m}^{(k)}$ sent by client $m \in M$ to form the base $w_{b,g}^{(k+1)}$ and head

$w_{h,g}^{(k+1)}$ of the global model $w_g^{(k+1)}$ for the next round:

$$w_{b,g}^{(k+1)} = \sum_{m=1}^{M} \gamma_{b,m} \times w_{b,m}^{(k)} \tag{3.9}$$

$$w_{h,g}^{(k+1)} = \sum_{m=1}^{M} \gamma_{h,m} \times w_{H1,m}^{(k)} \tag{3.10}$$

After aggregation, the next communication round begins, and the model is sent again.

Chapter 4

Implementation

This chapter will specifically introduce the details of the algorithm's implementation in code, focusing on three main parts: the generation of the rebalanced dataset, the server-side federated learning component, and the client-side local training. Additionally, details of the communication aspects necessary for deployment on the testbed will be introduced. The simulation part of the code primarily utilizes the Python machine learning library, `PyTorch`.

The main framework of the code implementation is based on the open-source repository – Personalized Federated Learning Platform [7], which includes implementations of various federated learning and personalized federated learning algorithms published in well-known conferences and journals, such as FedAvg [29], FedProx [28], FedDyn [10], etc. This repository facilitates experimentation and comparison and is continuously updated. The latest update includes the GPFL algorithm, published in ICCV 2023 in October 2023 [39].

4.1 Rebalancing Dataset Creating Algorithm

The original dataset is partitioned using a Dirichlet distribution, with code implementation mainly referencing [38]. The generation of the rebalanced data occurs after each client has been allocated their local simulation dataset. The local client simulation datasets are loaded using `PyTorch`'s `DataLoader` class. This allows for the counting of each class's data volume e_i as mentioned in the algorithm, using the labels of all samples contained in the `DataLoader`.

Since the data labels are represented numerically (for example, in a 10-class dataset, labels are represented by numbers 0-9), counting each class can be simply implemented through a loop. The creation of new samples through data augmentation is achieved using `PyTorch`'s `torchvision.transforms()`.

As the experimental part of the thesis primarily uses the Cifar10 and FMNIST image datasets, the `AutoAugment()` function provided by `PyTorch` is used for data augmentation. The effectiveness of this fucntion is also demonstrated in the paper [17].

For the part involving random sample extraction, the random.sample function from Python's built-in random library is used. This involves randomly selecting sample indices and then extracting the corresponding data and labels from the dataloader.

Regarding data saving, an empty dataset is initialized to store data, as mentioned earlier. In actual implementation, two numpy arrays provided by numpy are initialized to separately save data and their corresponding labels. Finally, the data is saved as an `npz` file using the `savez_compressed()` function provided by numpy.

4.2 Implementation Details of FedReb Server

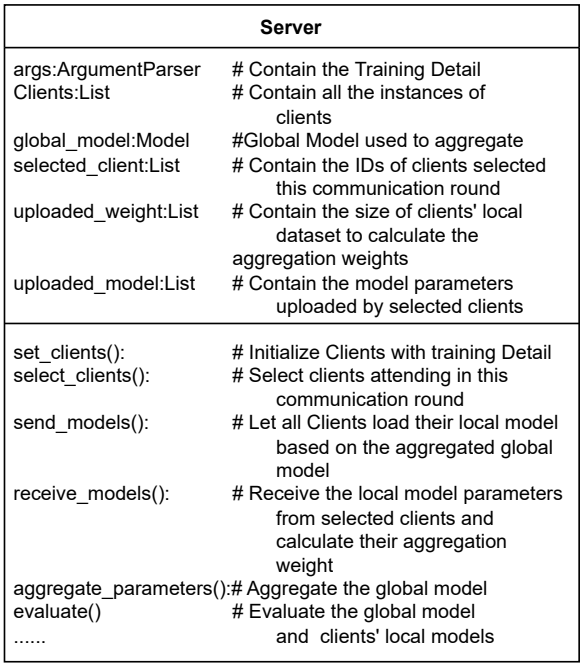


Figure 4.1: **Key variables and function of Class Server**

For the server part of the simulation federated learning, different federated learning algorithms are implemented as subclasses of the **Server** class. As shown in Figure 4.1, the `Server` superclass contains specific training process parameters as class variables for direct access and includes basic class functions such as running local evaluation and aggregation.

During instantiation, as there is a need to compare the accuracy of global models of different algorithms in the experimental part, the global model is initialized as a complete model. K client instances are initialized, and this model's structure and parameters are copied to the clients using Python's `copy.deepcopy()` function, instead of only initializing a base or head part of the model that cannot perform local prediction on the server side.

In the actual aggregation part, if there is no special requirement for aggregation weights like in this scheme, the entire model of all clients participating in the current round is used for global model aggregation after the simulation clients have completed training, following the FedAvg approach. In the local simulation code, the size of the user dataset is not repeatedly sent to the server

in each communication round, as the server can directly access the dataset size by calling the client's class variable.

However, the aggregation weights $\gamma_{b,m}$ and $\gamma_{h,m}$ need to be recalculated in each communication round due to different clients being selected. The aggregated model is saved as a variable of the simulation server and is also saved as a `pth` file used by `PyTorch` for saving model parameters, to facilitate the continuation of previous training in case of unexpected simulation interruptions.

4.3 Implementation Details of FedReb Client

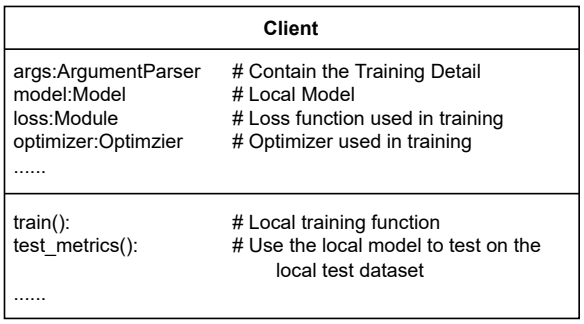


Figure 4.2: **Key variables and function of Class Client**

In the implementation of the client part of the algorithm, different federated learning algorithms are implemented as subclasses of the `Client` class. Similar to the `Server` class, as shown in Figure 4.2, the `Client` class contains class variables related to training details and basic functions such as local testing and loading global model parameters into the local model.

During instantiation, the user loads the original dataset and the corresponding rebalanced dataset according to their assigned ID, initializes a model according to the model provided by the
textttServer, and then copies the model's head as the local personalized layer head $H2$ on top of this model using Python's `copy.deepcopy()` function.

The original model's head is the head $H1$ mentioned earlier for aggregation on the server. Since it is actually the original local model's head, it can directly call the Server superclass's aggregation function without writing additional aggregation functions for the algorithm. Subsequently, the loss function l and two optimizers opt_1 and opt_2 are initialized, specifying their updated model parts as "`model.base+Head H2`" and "`model.base+model.head`" during initialization.

4.4 Communication on the Testbed

Considering network instability during communication or poor performance of the experimental equipment leading to interruption or freezing of the experimental process, a more mature communication protocol, MQTT, is used when deploying on the testbed. MQTT is a communication protocol designed for IoT field applications, and due to its subscription and publishing mechanism,

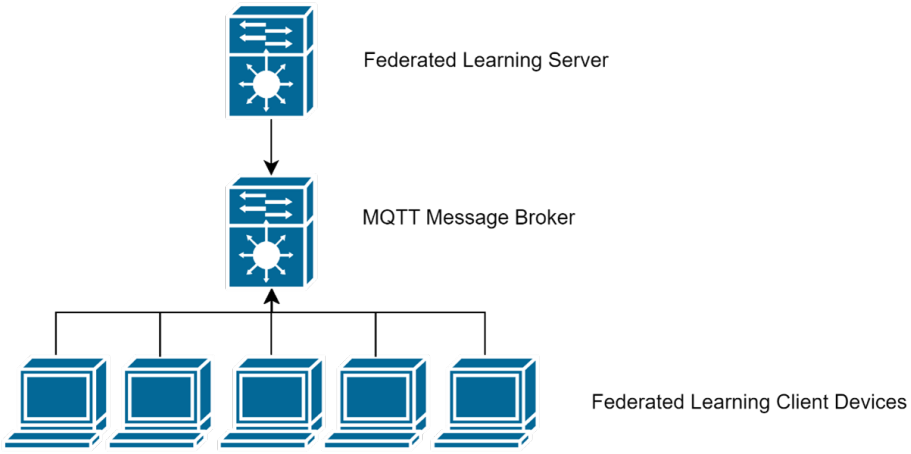


Figure 4.3: **The Communication Scheme of MQTT**

as shown in Figure 4.3, it requires a message broker server to handle all the messages sent by clients on various topics [30].

In the testbed experiment, the message broker used is a personal computer running the Windows 11 operating system. `Eclipse Mosquitto` [14], an open-source MQTT message broker software, needs to be installed on the personal computer and run as a background system service so that all relevant clients can connect to the message broker server (personal computer) through the local area network's IP address or the service URL provided by the server to transmit messages through the message intermediary. Figure 4.4 shows the background operation status of Mosquitto.

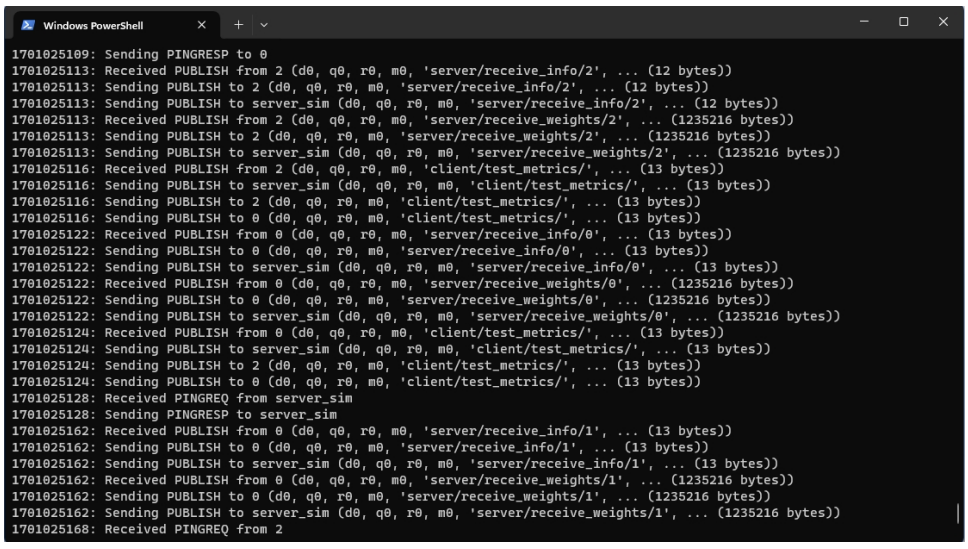


Figure 4.4: **Message log of MQTT message broker Mosquitto**

In the communication framework mentioned earlier, both the simulation server

and clients are actually clients relative to the message broker server. When deployed on the testbed, the Python library `paho_mqtt` is used to implement activities as a client.

The specific implementation is to instantiate a `paho.mqtt.client` client in the Python code and connect it to the message broker server. This essentially establishes a communication channel between the user and the message intermediary. After establishing the connection, the client needs to specify the topics they want to subscribe to in order to receive messages published on those topics.

The MQTT client has several very important class functions or member functions, namely the `on_connect()` function, the `on_message()` function, and the publish function. The rest of the text will refer to this instance as `mqtt_client`.

The publish function is relatively easy to understand; the `mqtt_client` can publish messages on the topics it subscribes to using the `mqtt_client.publish()` function. The `on_connect()` function and the `on_message()` function are two response functions.

The `on_connect()` function is passively triggered when the **mqtt_client** establishes a connection with the message broker using the `mqtt_client.connect()` function. The `on_message()` function is passively triggered when the **mqtt_client** receives messages published on the topics it subscribes to and is used to process messages. In implementing code that can communicate on the testbed, the communication part mainly involves the federated learning server and clients sending and aggregating models. Next, the 9 topics used in the implementation and their functions will be explained.

It should be noted that the '#' in the topics is a wildcard in MQTT, meaning subscribing to all subsequent levels of topics under the topic. For example, subscribing to "server/receive_info/#" would subscribe to "server/receive_info/0", "server/receive_info/1", and other topics.

Here are the details of the nine MQTT topics used for communication in the testbed setup:

1. **"client/train_detail/"**: This topic is primarily used for the federated learning server to send details of the experiment. In practice, these details are typically stored in an instance of Python's command-line parsing class `argparse.ArgumentParser`, whose base class is `NameSpace`, a class that can hold various types of variables as class variables.

   Since MQTT cannot directly send a class as a message, it's necessary to use the `json.dumps()` function from the `JSON` library to convert Python's key-value pair data collections or arrays into binary data that MQTT can send. This is done by first converting the training details `args` into a dictionary, then sending it through `json.dumps()`.

   It's important to note that when the server sends these experimental details, the message's "`retained`" flag should be set to True. This ensures that new clients subscribing to this topic after the message is sent can still receive it as the latest message.

   On the client side of federated learning, the `on_message()` function can read this binary message using the `json.loads()` function from the `JSON` library and reload it into the `NameSpace` class. This approach simplifies the process of converting simulation code into code deployable on a testbed, particularly in terms of reading training details, requiring minimal

adjustments to the simulation code.

2. **"client/connected/"**: This topic is mainly for the federated learning server to confirm how many and which clients have established a connection. Each experimental device simulates multiple clients and assigns them IDs. After establishing a connection and receiving the training details from the previous topic, clients actively send a dictionary composed of a list of locally simulated client IDs and a Boolean value representing the connection status to this topic using the `mqtt_client.publish()` function.

   Since **mqtt_client** cannot directly send dictionaries, the `json.dumps()` function from the `JSON` library is used to convert the list into binary data for transmission. Additionally, if a client loses connection, it will send a message on this topic using the Last Will and Testament (LWT) feature, changing the Boolean value for connection status to false.

   On the server side, the `on_message()` function will load the message content into a list using `json.loads()` to confirm the IDs of clients that have connected or lost connection, and update a local dictionary `client_stat` that maintains client status, changing the Boolean value of the "`connect`" key for each client iD to indicate whether the client is connected.

3. **"server/send_weights/"**: This topic is mainly used by the server to send the global model. Since the server model and the local model have the same structure in the algorithm, and to avoid exceeding MQTT's data size limit for messages, the server only sends the global model's parameter dictionary, generated by calling `global_model.state_dict()`.

   For transmission, the parameters are first saved into an instance of `io.BytesIO` from the IO library using `PyTorch`'s `torch.save()` function. The instance of `io.BytesIO` acts as a virtual file containing binary content of any file format, which is then sent through `mqtt_client`.

   This approach is used because the `state_dicts()` output from a `PyTorch` model is not a dictionary that can be converted into binary data by `JSON`, as it contains tensors of the model and must be sent in a binary file format. On the client side, the `on_message()` function handles this message by loading it through an instance of `io.BytesIO`, and then the local model uses the `model.load_state_dict()` function to load it as a file containing state dicts into the local model.

4. **"server/receive_info/#"**: This topic is primarily used by the federated learning server to confirm the current status of clients. The server subscribes to this topic and its subsequent levels, while federated learning clients only subscribe to the topic corresponding to the client ID simulated on their device, such as **"server/receive_info/0"**.

   In each communication round, if training is completed, the current client and the number of the training round are sent under this topic. After receiving this, the server's `on_message()` function updates the value of the `current_round` key for the corresponding client ID in the client status dictionary `client_stat`. When the `current_round` value for all clients selected in the current round matches the current communication round, the federated learning server begins aggregation for that round.

5. **"server/receive_weights/#"**: This topic is used by the federated learning server to receive model parameters from clients with corresponding IDs. The approach of transmitting parameters is the same as mentioned in the **"server/send_weights/"** topic, implemented through an instance of the `io.BytesIO` class and **PyTorch**'s "save()" and `load_state_dict()` functions for `PyTorch` models.

6. **"server/current_round/"**: This topic is used by the federated learning server to send the current communication round number k, allowing all clients to synchronize with the current communication round. Along with the **"server/receive_info/#"** topic, it helps determine the communication round number of the clients participating in training to aggregate the global model. The message can have its `retained` flag set to True to preserve the latest message, so clients can confirm the current communication round after reconnecting and read the locally downloaded model.

7. **"client/selected/"**: This topic is used by the federated learning server to send the list of clients selected for the current communication round. The IDs of the selected clients are stored in a list, which can be converted into binary data using the `json.dumps()` function from the `JSON` library and sent through `mqtt_client`. The `retained` flag can also be set to True to preserve the latest message, allowing federated learning clients to confirm upon reconnection whether the locally simulated client is participating in the current round of training.

   Clients handle this in the `on_message()` function by loading the data with `json.loads()` and comparing the IDs of the locally simulated clients against the list to determine participation in the current round of training.

8. **"client/test_metrics/"**: During the server testing phase, this topic is used to receive local test results from clients for calculating metrics such as average accuracy. The data, stored in a list, is transmitted and read using the `JSON` functions mentioned earlier, with decimals retained to four decimal places.

9. **"train/end/"**: This topic is used by the federated learning server to send a command to end training. After receiving this command, clients will confirm whether this instruction is True after completing local training to end the local simulation.

Chapter 5

Experiments

In this chapter, a brief introduction to the baseline algorithms used for comparison will first be provided. This will be followed by a description of the datasets used and the experimental configurations implemented. Subsequently, the chapter will explore the optimal configuration for achieving the best performance with the FedReb algorithm. This exploration will include an analysis of the impact of two key factors: the creation method of the balanced dataset, and the number of personalized head layers and aggregation weights during the algorithm's aggregation process. Recommendations for configurations that yield the best performance will be provided. Finally, FedReb will be compared with other baseline algorithms under the optimal configuration, and the advantages of the FedReb algorithm over these alternatives will be analyzed.

5.1 Experiment Baselines

The experiments mainly compare two types of federated learning approaches. The first type includes traditional federated learning schemes such as FedAvg [29], FedProx [28], and FedDyn [10]. The second type involves personalized federated learning schemes like Ditto [27], FedPer [12], FedRep [16], FedBABU [31], and FedROD [15]. Among these, FedPer, FedRep, FedBABU, and FedROD are algorithms that, similar to this thesis, implement personalized federated learning through parameter decoupling.

1. **FedAvg (Federated Averaging)** [29]: As mentioned earlier, it is the earliest federated learning algorithm. It works by clients learning local models and then aggregating these models on a centralized federated learning server. Compared to traditional centralized machine learning, it enhances user data privacy and reduces communication overhead. However, it performs poorly in the face of non-IID data distribution.

2. **FedProx (Federated Proximity)** [28]: This algorithm is designed based on FedAvg to address heterogeneity issues in federated learning, including data heterogeneity caused by non-IID data distribution and system heterogeneity due to different computational and communication capabilities of participating devices. The algorithm introduces an additional proximal term in the local optimization problem to aid in the stability

and improvement of the algorithm's convergence, especially in the face of data heterogeneity caused by non-IID data distribution.

3. **FedDyn (Federated Dynamics)** [10]: This is another traditional federated learning algorithm designed to address problems caused by data heterogeneity. The algorithm introduces a dynamic regularizer in the local loss function to align local model updates more closely with the global model's optimal solution, rather than just the local model's optimal solution, reducing the impact of data heterogeneity on the convergence speed and accuracy of the federated learning global model.

4. **Ditto** [27]: A personalized federated learning scheme implemented through regularization. It aims to strike a balance between a single global model in federated learning and a personalized model trained locally without any federated learning scheme.

   During training, clients actually have two models: a local model and a model that continuously participates in traditional federated learning. It achieves personalization by optimizing the local model to keep it close to the federated learning global model.

5. **FedPer (Federated Personalization)** [12]: A personalized federated learning algorithm and one of the main reference algorithms for the thesis, implemented through parameter decoupling. The model is split into a base shared with the federated learning server and a head retained locally for personalization, allowing the model to accommodate different clients' personalized needs more effectively.

   During the learning process, local training trains the complete base + head part of the model, but since only the base part of each client is aggregated, it is impossible to aggregate a global model capable of completing the task.

6. **FedRep (Federated Representation Learning)** [16]: A personalized federated learning algorithm that also decouples parameters, splitting the model into a base shared with the federated learning server and a locally retained head.

   The aggregated base is considered to have learned the global feature representation of each client, while the head, although not called a personalization layer like in FedPer, is mainly used for local personalization. The main difference from FedPer is that during local training, the user's head is trained for multiple rounds to meet personalization needs, and then the aggregated base is trained for only one round to learn the local feature representation.

7. **FedBABU (Federated Body and Batchnorm Update)** [31]: Also a personalized federated learning scheme implemented through data decoupling, it similarly splits the model into a base and a head. Unlike FedPer, during training, only the base part is trained and aggregated, while the head part is fine-tuned with local data for local testing. This approach primarily enhances the learning capability of feature representation for image classification tasks in personalized federated learning frameworks.

8. **FedROD (Federated Robust Decoupling)** [15]: Another personalized federated learning scheme based on data decoupling and another main reference algorithm for the thesis. It splits the model into a base and two identical heads, one for aggregation and one for local personalization, and trains the two heads with different loss functions.

   The complete model base and the head used for aggregation are trained together using a balanced softmax loss function to calculate gradients. The other head, used for local personalization, is trained separately using a common cross-entropy loss function. The aim of this algorithm is to simultaneously obtain a better global model and a personalized local model for clients. However, the paper only tested scenarios with 5 out of 20 clients participating in training, lacking scalability.

5.2 Dataset

This section introduces two commonly used datasets in the experiments: Cifar10 and Fashion-MNIST (FMNIST).

5.2.1 Cifar10

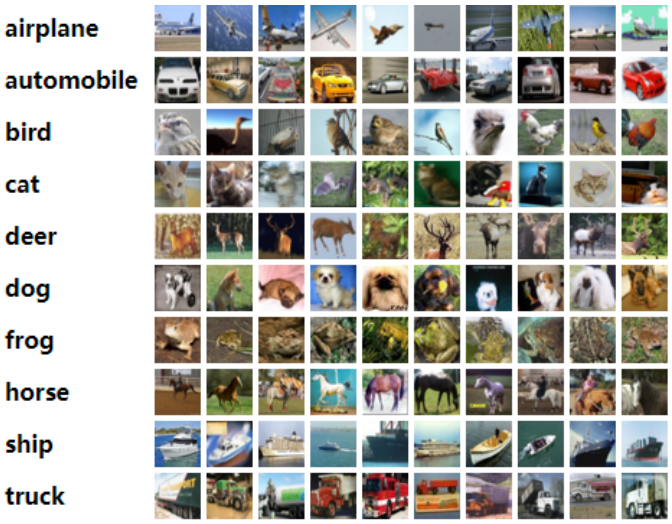


Figure 5.1: **Samples of data in Cifar10 [11]**

The Cifar10 dataset is a popular dataset for machine learning and computer vision, consisting of 60,000 color images with a resolution of 32x32, divided into 10 different classes. Each class, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck, contains 6,000 images [24]. Figure 5.1 displays several sample images from each class.

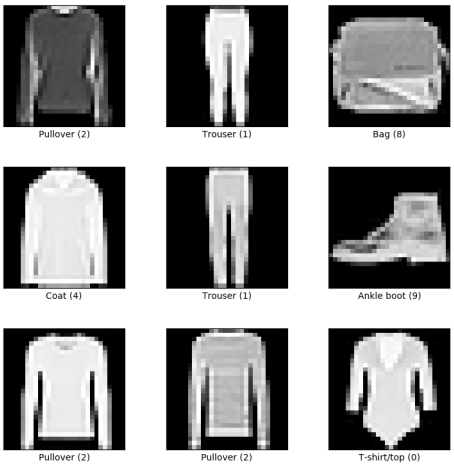


Figure 5.2: **samples of data in Fashion-MNIST [37]**

5.2.2 Fashion-MNIST (FMNIST)

The Fashion-MNIST dataset comprises 70,000 grayscale images of 28x28 resolution. It is categorized into 10 different types of fashion products, including coat, trousers, shirts, sneakers, bags, etc [36]. Figure 5.2 shows some samples from the dataset.

5.3 Experimental Configuration

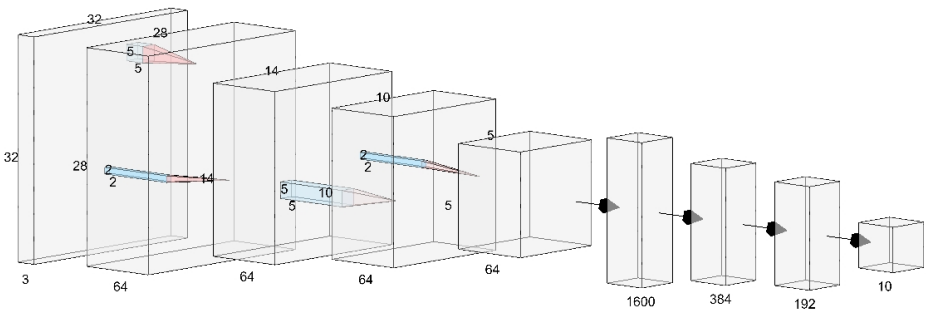


Figure 5.3: **ConvNet Architecture used to learn Cifar10 dataset**

The experimental model adopts the ConvNet used in the FedDyn and FedROD paper [10][15], as shown in Figure 5.3, consisting of two convolutional

layers, one pooling layer after each convolutional layer, and three fully connected layers, with the activation function of the fully connected layers being the ReLU function.

The experimental datasets used are Cifar10 and FMNIST. The datasets' training and test sets are first mixed and then distributed using a Dirichlet distribution with α values of 0.1 and 0.5. Of the distributed data, 75% is used as the training set for the clients, and 25% as their test set. All clients' test sets are combined to form the global model's test set.

For client participation rates in each communication round, two approaches are tested: selecting 5 out of 20 clients and 20 out of 100 clients per communication round. The communication rounds are set to 100, with 5 local training rounds for clients, a batch size of 20, an optimizer learning rate of 0.01, and a momentum of 0.9.

The data provided in the tables are the averages of five runs with the same experimental configuration. In each run, the highest global model accuracy (best global model accuracy) and every client accuracy (best average client accuracy) out of 100 rounds are recorded and then calculate their average values.

The global model and all client local models are tested after the federated learning server sends out the global model. The average client accuracy is actually calculated by summing the number of correctly predicted samples and the total number of samples in the test set, then dividing the two. The global model is tested using the combined test sets of all clients.

In comparison algorithms, personalized federated learning algorithms like FedPer, FedRep, and FedBABU do not have a complete global model for prediction. The entire local models of clients are aggregated during the aggregation process, and after distributing the global model, clients use parts of the parameters according to their algorithms.

In traditional federated learning approaches, local model parameters are identical to global model parameters. Therefore, for these traditional federated learning approaches, the accuracy of the model distributed to the local and tested on the local test set (average client accuracy) is mostly the same as the global model accuracy in most cases.

The additional hyperparameters introduced by different algorithms:

1. FedAvg: A traditional federated learning algorithm using the above experimental configuration, with no additional hyperparameters.

2. FedProx: Due to the additional proximal term introduced in optimization, there is an extra hyperparameter, mu, representing the strength of the proximal term. In the experiments, **mu** is set to 0.001, a commonly used option in the original paper's code [1], and also used in FedROD [15]for replication and comparison.

3. FedDyn: The strength of the dynamic regularizer, **alpha**, in the algorithm is set to 0.01 in the experiments, also referring to the source code [3] and the experimental configuration in FedROD.

4. Ditto: Since there are actually two models at the client's local: a federated learning model and a local personalized model, the training rounds of the local personalized model become an extra hyperparameter, termed personal local step **pls**. It is set to 5 in the experiments, the same as

other algorithms' local training rounds. The federated learning model's local training rounds are also 5.

5. FedPer: No extra parameters, but since the original algorithm theoretically only has a base for the global model and cannot be used for testing, in practical testing, the entire models of clients are aggregated as the global model for testing global model accuracy.

6. FedRep: The training rounds of the local personalized model's head are an extra hyperparameter, also represented by the variable personal local step **pls**, set to 5 in the experiments. The training rounds of the model base are as described in the original paper and source code, only one round [2].

7. FedBABU: Since only the base of the local model is trained during local model training. And the head part is trained before testing, there is an extra hyperparameter, the number of fine tuning rounds of the head part **fine_tune**, set to 10 in the experiments.

8. FedROD: No extra hyperparameters, although it introduces an additional softmax balance loss function and optimizer, the loss function has no extra parameters, and the optimizer's learning rate is the same.

5.4 Impact of Size and Creation Methods of the Rebalanced Dataset

In generating the rebalanced dataset, the experiment selected four values as thresholds t: **the maximum, median, mean, and the second minimum value** of the class sample sizes that are not zero. The effects of different thresholds were tested. Figure 5.4 shows the data distributions of rebalanced datasets created under different thresholds t.

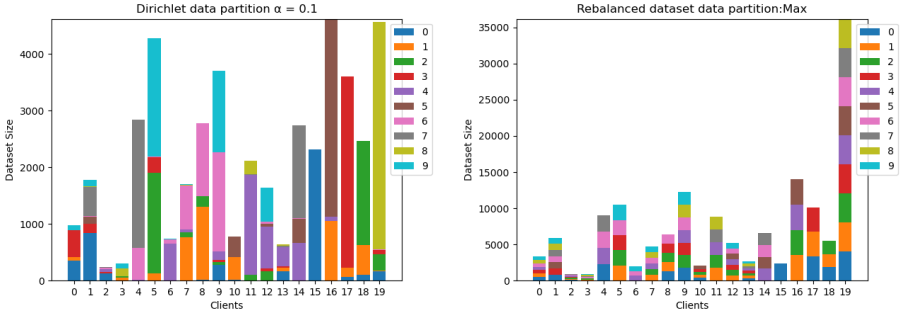
Two methods were used to generate the data through augmentation: one is the **AutoAugmentation()** function provided by the `PyTorch` library, and the other is a common, simple **5-step data augmentation** process.

The reason for choosing the second minimum class sample size as one of the threshold values t is that using the minimum value as t would not invoke the data augmentation method. All other class data would be reduced to the minimum class size, making it impossible to assess the impact of data augmentation on the rebalanced dataset.

Another reason is that the client data allocated through the Dirichlet distribution sometimes results in classes with only one sample. In such cases, all other class data in the rebalanced dataset would be reduced to one, which is too small to significantly impact the model. However, even selecting the second minimum value as t might still result in another class with only one sample. This threshold setting aims to avoid the issue of excessively small rebalanced datasets.The data distribution through all the client under different threshold t are shown in Figure 5.4

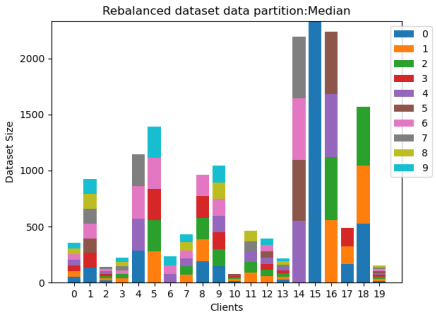
5.4.1 The Impact of Augmentation Methods

The `AutoAugmentation()` function in `PyTorch` includes over 20 different data augmentation operations, with preset enhancement strategies for datasets like

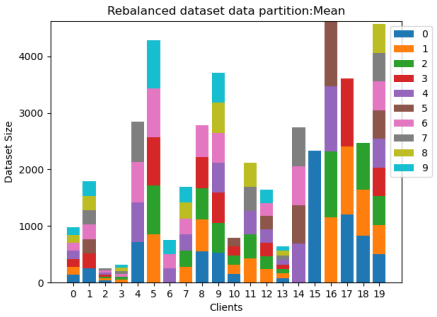


(a) **Original**

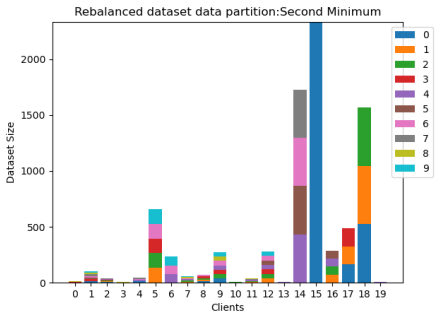
(b) **Maximum**



(c) **Median**



(d) **Mean**



(e) **Second Minimum**

Figure 5.4: **Clients' data distributions of Rebalanced Dataset under different threshold t.**

Cifar10, IMAGNET, and SVHN [17]. The 5-step PyTorch data augmentation process used in the experiment consists of random horizontal flipping, random cropping, random rotation, color jitter, and random affine. For FMNIST, which consists of grayscale images, the color jitter step is omitted. Table 5.1 shows the impact of different settings on the Cifar10 dataset, with 5 out of 20 clients participating per round and a Dirichlet α value of 0.1. The number of head layers of the model used in these experiments is 2. The main differences are reflected in the global model accuracy, with less noticeable differences in average accuracy.

From Table 5.1, it is evident that the rebalanced datasets generated using

Table 5.1: **Table shows the influence of different Data Augmentation methods on the algorithm.**

Aug Method	Threshold t	Best Global Acc	Best Average Acc
AutoAug	Max	53.27%	87.81%
	Median	52.58%	89.77%
	Mean	55.26%	88.90%
	SecMin	53.57%	90.08%
5-step	Max	59.37%	90.29%
	Median	55.84%	89.64%
	Mean	60.40%	90.25%
	Sec-Min	54.66%	90.58%

the simple 5-step data augmentation scheme consistently enable the FedReb algorithm to achieve higher global model accuracy.

This indicates that in generating rebalanced datasets, a more complex data augmentation scheme is not necessarily better, and the generation of rebalanced datasets does not rely on how new data is generated through data augmentation to supplement classes with fewer samples than threshold t. This may be caused by the `AutoAugmentation()` method introducing too much noise to the ConvNet model, as shown in Figure 5.5. While 5-step augmentation only introduces slight changes in the images, as shown in Figure 5.6. ConvNet is too simple for image classification.

5.4.2 The Impact of Threshold t

Table 5.2: **Table shows the influence of different selection of threshold t to the algorithm. When the t is the maximum or the mean value of the quantities of the classes that are non-zero, the algorithm has its best performance.**

Threshold t	Best Global Acc	Best Average Acc
Max	59.37%	90.29%
Median	56.90%	89.94%
Mean	60.40%	90.25%
Sec-Min	55.69%	90.38%

Table 5.2 isolates the results of rebalanced datasets generated through the 5-step data augmentation from Table 5.1, aiming to compare the impact of different threshold values t on the algorithm. From Table 5.2, it is observed that the algorithm performs best when the threshold t is set to either the maximum or average value of all non-zero classes. Compared with Table 5.1, similar results are seen when using `AutoAugmentation()`, with the highest performance when the threshold is the mean value of all non-zero classes.

However, comparing with tables 5.5 and 5.6 that compare various algorithms, it is found that even when the threshold t is set to the second minimum value, FedReb still achieves a better global model than other personalized federated learning algorithms, except for FedROD, particularly the FedPer algorithm,

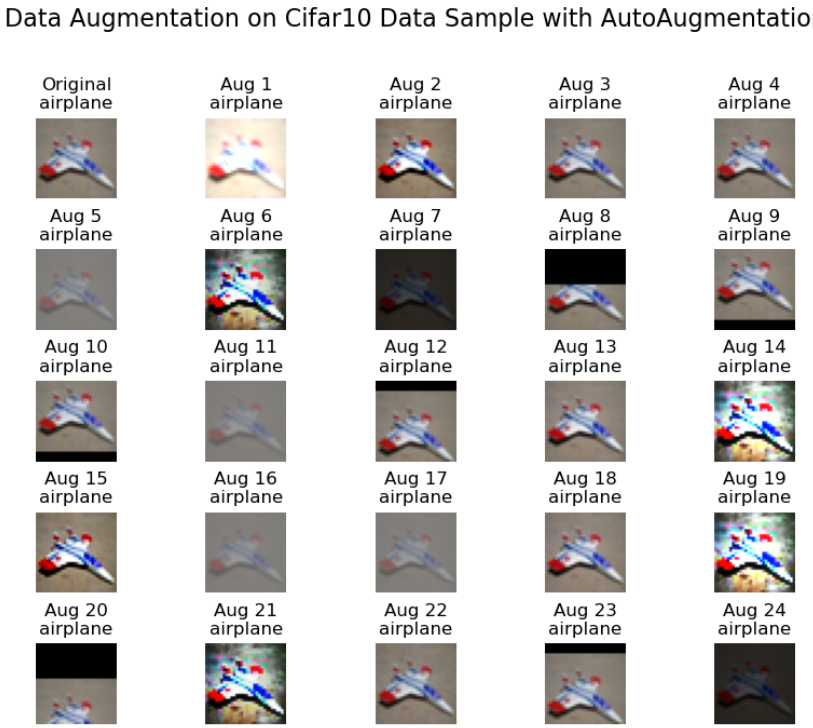


Figure 5.5: **Data Augmentation on Cifar10 Data Sample with AutoAugmentation.**

which was a main reference for this thesis.

This shows that the FedReb algorithm, by using the rebalanced dataset architecture alone, can improve the accuracy of the global model compared to other personalized learning algorithms that do not specifically optimize the global model. When an appropriate threshold t and data augmentation scheme are selected, it can achieve performance close to FedROD in scenarios with a small number of clients, and surpass FedROD in terms of global model accuracy and average accuracy in scenarios with a larger number of clients.

5.4.3 Recommend Configuration

From the experimental results, for Cifar10 and Mnist, a simple 5-step data augmentation strategy can be chosen. The appropriate threshold t should be the mean value of all non-zero classes. Using either the maximum value or the mean value of all non-zero classes as t to generate the rebalanced dataset can enable FedReb to achieve high global model accuracy and average accuracy, with similar results between the two. The dataset generated using the mean value is smaller in size and requires less training overhead.

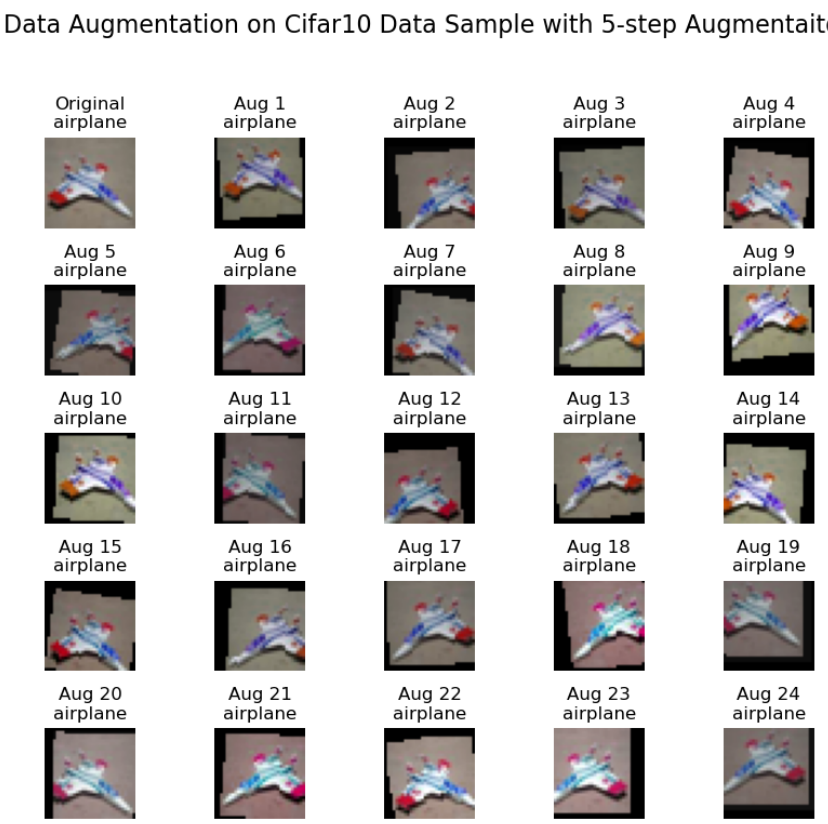


Figure 5.6: **Data Augmentation on Cifar10 Data Sample with 5-step simple augmentation.**

5.5 Impact of Head Layer Depth and Choice of Aggregation Weights

The impact of the number of personalized head layers has actually been mentioned in the FedPer paper[12], where it was concluded that the impact of personalized layer depth on the results is very minimal. This section similarly demonstrates the effect of the number of layers designated as personalized (head) on the experimental results. Additionally, a comparison was made between the performance of FedReb when aggregating the global model using separate aggregation for base and head, versus the traditional federated learning aggregation approach, to select the appropriate aggregation scheme.

5.5.1 The Impact of Head Layers Depth

The experiment used the Cifar10 dataset, with 5 out of 20 clients participating per communication round, and a Dirichlet α of 0.1.

From Table 5.3, it is evident that the number of personalized head layers in the model has a minimal impact on both the global model accuracy and average

Table 5.3: **Table shows the influence of different numbers of head layers and aggregation weight on the algorithm. The threshold t used in these experiments is the mean value of the class sample sizes that are not zero.**

Layers	Agg Weight	Best Global Acc	Best Average Acc
Origin	3	59.72%	90.19%
	2	57.68%	89.76%
	1	59.05%	89.56%
Split	3	59.90%	89.82%
	2	60.40%	90.25%
	1	59.18%	89.64%

accuracy of the FedReb algorithm, similar to the conclusions in the FedPer paper regarding the minimal impact of different personalized layer depths. The only instance where a decrease in global model accuracy was observed was when the model's head had two layers and the traditional federated learning aggregation approach was chosen.

5.5.2 The Impact of Aggregation Weights

Table 5.4: **Table shows the influence of the aggregation weight to the algorithm. The number of head layers of the model used in these experiments is 2.**

Agg Weight	Threshold t	Best Global Acc	Best Average Acc
Origin	Max	57.61%	89.62%
	Median	56.90%	89.94%
	Mean	57.68%	89.76%
	Sec-Min	55.49%	90.28%
Split	Max	59.37%	90.29%
	Median	58.14%	90.00%
	Mean	60.40%	90.25%
	Sec-Min	54.66%	90.58%

Comparing with the data in Table 5.4, which shows the impact of different threshold values t and aggregation schemes on FedReb, it is apparent that in most cases, FedReb performs better when the base and head are aggregated with different weights.

5.5.3 Recommand Configuration

From this, it can be inferred that for a scheme like FedReb that uses rebalanced datasets, it is more effective to use different weights for aggregating the base and head. Additionally, the number of layers in the model's head has little impact on the results under this aggregation scheme. Nevertheless, a head layer depth of two marginally enhances both the global accuracy and the average accuracy in the FedReb model.

5.6 Comparative Results

Table 5.5: **The table includes the performance of all comparison algorithms on the Cifar10 dataset. 'G' represents the Best Global Model Accuracy, and 'P' denotes the Best Client Average Accuracy. The data of FedReb are written in blue.**

Dataset	Cifar10							
joining rate	5 of 20				20 of 100			
α / Alg.	0.1		0.5		0.1		0.5	
	G	P	G	P	G	P	G	P
FedAvg	54.92%	54.92%	65.47%	65.47%	49.50%	49.50%	60.20%	60.20%
FedProx	55.41%	55.41%	65.34%	65.34%	51.32%	51.32%	60.44%	60.44%
FedDyn	56.27%	56.27%	63.64%	63.64%	51.75%	51.75%	57.56%	57.56%
Ditto	55.91%	88.19%	65.94%	68.86%	49.39%	84.68%	60.82%	60.82%
FedPer	47.75%	90.40%	62.22%	75.86%	44.22%	87.93%	53.93%	67.49%
FedRep	41.20%	89.64%	55.78%	72.80%	36.25%	85.91%	43.72%	61.17%
FedBABU	53.25%	89.02%	58.57%	73.75%	39.42%	85.18%	49.65%	64.82%
FedROD	59.14%	89.62%	66.29%	76.71%	49.85%	88.38%	59.69%	73.94%
FedReb	60.40%	90.25%	65.96%	76.76%	57.70%	89.20%	62.15%	74.06%

Table 5.6: **The table includes the performance of all comparison algorithms on the Fashion-MNIST dataset. 'G' represents the Best Global Model Accuracy, and 'P' denotes the Best Client Average Accuracy. The data of FedReb are written in blue.**

Dataset	FMNIST							
joining rate	5 of 20				20 of 100			
α / Alg.	0.1		0.5		0.1		0.5	
	G	P	G	P	G	P	G	P
FedAvg	83.08%	83.08%	90.12%	90.12%	81.27%	81.27%	87.61%	87.61%
FedProx	83.91%	83.91%	90.50%	94.21%	80.65%	80.65%	87.47%	87.47%
FedDyn	85.03%	85.03%	89.59%	89.59%	84.08%	84.08%	87.44%	87.44%
Ditto	84.20%	97.81%	89.97%	91.33%	81.63%	96.21%	87.69%	87.69%
FedPer	72.66%	98.30%	89.36%	94.05%	80.06%	97.42%	87.28%	92.01%
FedRep	58.24%	98.13%	87.99%	93.01%	61.43%	96.59%	77.61%	88.22%
FedBABU	71.27%	97.84%	85.73%	92.19%	75.48%	95.34%	83.88%	87.88%
FedROD	86.17%	98.14%	90.84%	94.18%	80.01%	96.74%	87.32%	93.10%
FedReb	87.29%	98.07%	90.46%	93.84%	88.21%	97.58%	90.38%	94.18%

Tables 5.5 and 5.6 present comparisons between various algorithms and the algorithm developed in this thesis. The columns labeled 'G' and 'P' represent the best global model accuracy and the best average accuracy of the personal model, respectively. The "joining rate" indicates the selection of clients for each communication round, such as "5 of 20," meaning five clients are chosen out of twenty for that round's training. FedPer and FedROD arethe reference algorithms in this thesis. FedROD is a recent personalized federated learning

algorithm that enhances global model accuracy. Some parts of the analysis will primarily compare these algorithms.

5.6.1 Robustness Analysis

Table 5.5 shows performance on the Cifar10 dataset. Under the condition of 20 clients choosing 5 with a Dirichlet distribution of α 0.5, where client data distribution is relatively balanced, FedReb's global model accuracy is slightly lower than FedROD's, but its average accuracy is high, similar to other personalized federated learning algorithms. It's also observed that personalized federated learning generally has higher average accuracy than traditional federated learning.

When the Dirichlet α drops to 0.1, indicating an imbalanced data distribution, FedReb achieves higher global model accuracy than most personalized federated learning algorithms, even surpassing traditional federated learning algorithms. The average accuracy is close to other personalized federated learning algorithms, with only minor differences, but still higher than traditional federated learning. Notably, as the Dirichlet α decreases, the decline in FedReb's global model accuracy is only 5.56%, lower than most other personalized and traditional federated learning schemes, such as FedROD, which drops by 7.14%.

Comparing Table 5.6, on the FMNIST dataset, with 20 clients choosing 5 per round, FedReb shows similar performance. It's observed that under the "5 of 20" condition, although FedReb's global model accuracy and average accuracy are very good, the difference with the FedROD algorithm is not significant. However, when the Dirichlet α decreases, the loss in global model accuracy for FedReb remains the lowest among all algorithms, dropping only by 3.17%.

Similarly, comparing both datasets under the condition of 100 clients choosing 20, FedReb maintains a lower accuracy loss when the Dirichlet α decreases and data becomes more imbalanced, demonstrating strong robustness. This robustness is evident as the data becomes more imbalanced and heterogeneous with a lower Dirichlet α, resulting in smaller losses in global model accuracy and average accuracy compared to other algorithms.

5.6.2 Scalability Analysis

Comparing data from the Cifar10 table Table 5.5, as the number of clients increases from 20 to 100 and the number of clients participating in each round increases from 5 to 20, Table 5.5 shows that FedReb's global model accuracy and average accuracy are generally higher than all other comparison subjects.

It's observed that under these conditions, the previously excellent-performing FedROD algorithm's global model accuracy significantly decreases. With the same Dirichlet α value of 0.1, FedROD's global model accuracy on Cifar10 drops from 59.14% to 49.85%, lower than traditional federated learning algorithms like FedProx and FedDyn, despite its design to enhance global model accuracy.

However, at this point, the global accuracy of other personalized federated learning algorithms is also not as high as traditional federated learning algorithms. When the Dirichlet α is increased to 0.5, FedReb's global accuracy and average accuracy remain the highest, even surpassing traditional federated learning methods.

It's evident that while FedReb's performance in the 20 choose 5 scenario is not the best, comparable to FedROD, it still maintains excellent results when the scenario changes to 100 choose 20, leading in global model accuracy among all comparison subjects and having an average accuracy close to FedROD, higher than other algorithms. This phenomenon is also observed in tests on the FMNIST dataset.

Thus, it can be concluded that the FedReb algorithm, compared to other algorithms, especially the newer FedROD, demonstrates better scalability. Scalability is evident in the algorithm's ability to maintain high global accuracy and average accuracy even as the number of clients increases.

Chapter 6

Validation on TestBed

6.1 Implementation of TestBed

The primary purpose of TestBed is to study how federated learning algorithms can be implemented and tested in real-world scenarios. The content of the experimental results section is still mainly based on local simulation tests. TestBed consists of a personal computer running Windows 11, multiple Raspberry Pi 4 Model B units, and a wireless router.

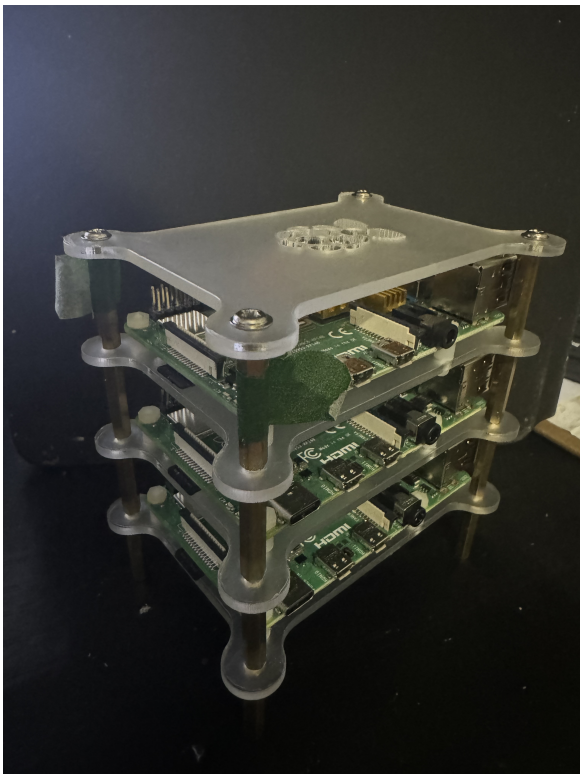


Figure 6.1: **Raspberry Pi 4 Model B units used in testbed**

Raspberry Pi is a type of small single-board computer (SBC), originally designed to promote basic computer science education through low-cost hardware and free software. It is commonly used in research and education fields [8].

The model used in TestBed is the Raspberry Pi 4 Model B, equipped with 8GB of RAM and a 1.5GHz ARM architecture central processing unit. It serves as a simulated federated learning client, where one Raspberry Pi can simulate multiple clients. However, if too many are simulated, the training speed is limited. During verification, no more than five clients are simulated. Additionally, since the Raspberry Pi only has a central processing unit and lacks a dedicated graphics processing unit or units specialized for training neural networks, training complex models can be very time-consuming. Therefore, simpler models like ConvNet and a two-layer fully connected DNN are used for deployment feasibility verification.

In TestBed, the personal computer acts as both a simulated federated learning server and an MQTT message broker. The computer is equipped with an NVIDIA RTX3080 graphics card, 64GB RAM, and an AMD Ryzen 9 5900HX central processor, operating at a frequency of 3.3GHz. To simultaneously send

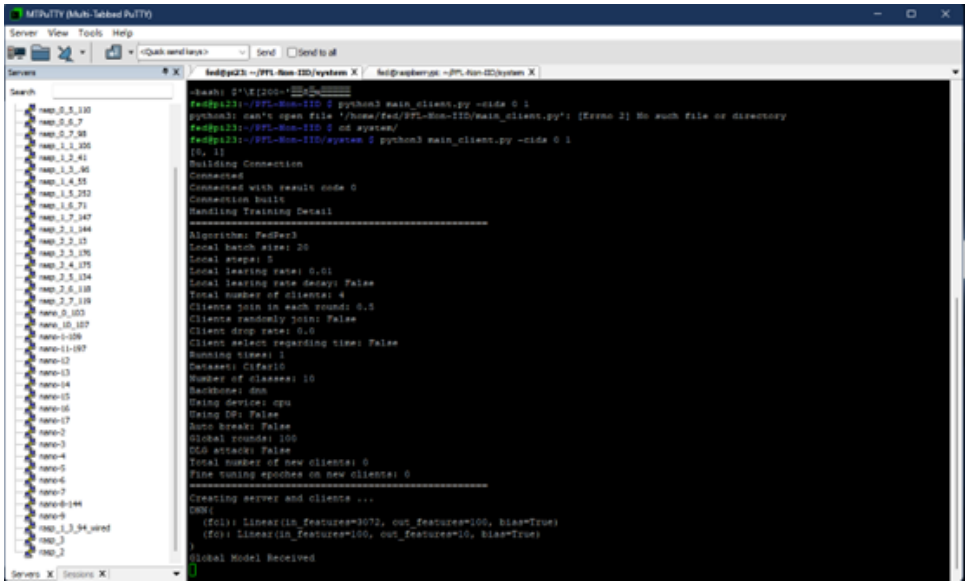


Figure 6.2: **MTPuTTY user interface when connecting with 2 Raspberry Pi units**

commands to all participating Raspberry Pis, MTPuTTY [6],as shown in Figure 6.2, is installed on the personal computer for running multiple SSH sessions and issuing the same command line instructions at the same time.

The experimental datasets and rebalanced datasets of clients are first generated on the simulated federated learning server (personal computer) and then uploaded to Google Drive. Although each Raspberry Pi must hold all clients' datasets, this facilitates adjusting the number of clients simulated on the Raspberry Pi. The total size of the compressed original datasets and the rebalanced datasets generated with threshold values t being the mean value of classes is 771.9MB. To facilitate unified data downloading on the Raspberry Pi units,

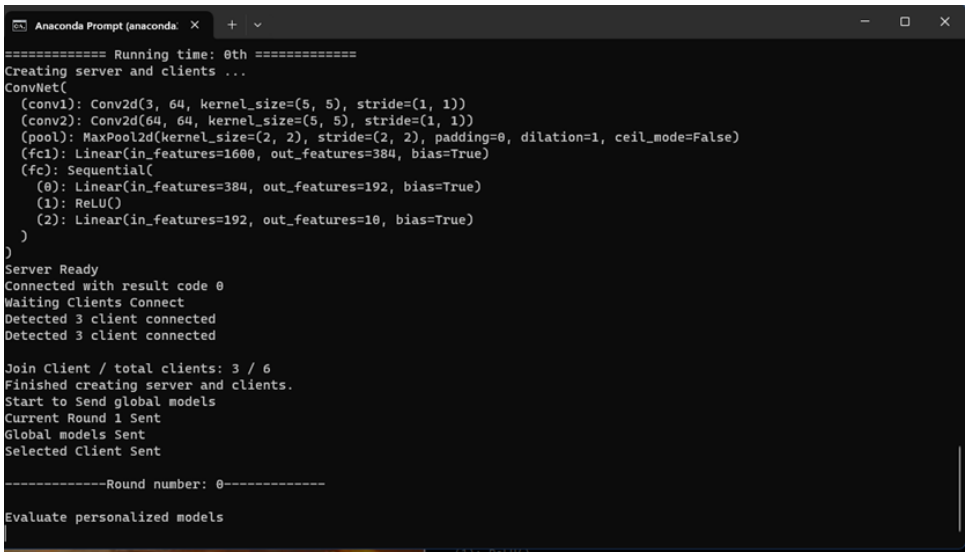


Figure 6.3: **Federated Learning Server running on the testbed**

`gdown` [4] is installed, allowing direct file downloads from Google Drive links via the command line. Combined with `MTPuTTY`, this enables the Raspberry Pis to uniformly obtain experimental data and decompress it locally. The code on TestBed is deployed via GitHub [5], and updates can also be pulled uniformly using `MTPuTTY` by sending commands simultaneously.

Due to performance limitations, TestBed only verified the scenario of up to four clients training for 100 rounds, which could be completed in full. By forcibly interrupting the training of clients, situations where clients disconnected during training were simulated, and the code's reconnection functionality was verified. However, due to the small number of clients, it is difficult to use this data for comparison with other algorithms. Therefore, the experimental results in chapter 5 are still primarily comparing simulations conducted on remote high-performance servers.

6.2 Result of Validation

In the feasibility verification part of the TestBed, due to the performance limitations of the Raspberry Pi, the model used is a DNN network composed of two fully connected layers, tested on the Cifar10 dataset.

This means the input of the DNN is $3 \times 32 \times 32$ (the size and dimension of Cifar10 images), with an intermediate layer of 100 and an output layer of 10, corresponding to the number of categories in Cifar10. The test data is derived from the simulation code, where the Cifar10 dataset is distributed to 20 clients using a Dirichlet distribution.

However, only 4 clients actually participate in the training, and these 4 clients are selected in every communication round. In other words, 4 clients are simulated using Raspberry Pi, with client IDs 0, 1, 2, 3, and they use the datasets of clients 0, 1, 2, 3 from the 20-client dataset.

The other test parameters are the same as those used in the chapter 5 experiments. The algorithms tested are only FedAvg and FedReb, to verify the feasibility of deploying the code on the TestBed.

After 100 training rounds, the results are as follows:

Alg	Global%	Average%
FedAvg	38.64%	54.96%
FedReb	39.72%	55.98%

The data in the table shows that there is no significant difference in performance between the FedReb and FedAvg algorithms. The most significant factor seems to be the model itself. A two-layer fully connected DNN model, when trained directly on the complete Cifar10 dataset, achieves an accuracy of around 37%.

Limited by the device, a ConvNet composed of two convolutional layers takes more than ten times longer per training round compared to the DNN. Therefore, a model without convolutional computations had to be chosen. To further verify the results of the algorithm, testing on devices specifically designed for convolutional computations would be necessary.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Now, through the content of this thesis, answers to the research questions posed at the beginning can be provided :

Secondary Research Questions:

1. **What causes data heterogeneity problems in federated learning?**

   In Chapter 2, it is explained that data heterogeneity in federated learning is caused by client drift.

2. **What are the solutions to the data heterogeneity problem in federated learning?**

   In Section 2.3 of Chapter 2, a research direction is discussed that aims to solve the data heterogeneity problem in federated learning while also addressing the lack of personalization in federated learning algorithms: personalized federated learning.

3. **What are the limitations of these solutions?**

   Personalized federated learning no longer aims to aggregate a global model that performs well across all clients. Instead, it focuses on training models on clients that meet their own task requirements or data distributions. However, in this process, the global model aggregated by the federated learning server often serves as an intermediate product and lacks the capability to complete tasks or test data, losing the potential of federated learning to form a generalized global model by learning the characteristics of different users' data.

4. **What approaches can be used to design algorithms that solve the problem of data heterogeneity?**

   Various approaches in personalized federated learning can effectively solve the problem of data heterogeneity, but they lack a well-performing global model. The FedROD paper proposes a personalized learning solution

based on parameter decoupling, which addresses personalization and data heterogeneity issues in federated learning while also obtaining a well-performing global model [15]. Both FedROD [15] and another algorithm, FedPer [12], which also implements personalized federated learning through parameter decoupling, address data heterogeneity issues and overcome the lack of a generalized global model in personalized federated learning. Therefore, designing a new personalized federated learning algorithm through parameter decoupling is an effective approach.

Main Research Question:

How to solve the performance degradation caused by data heterogeneity in federated learning?

A novel personalized federated learning scheme based on parameter decoupling is designed , Federated Learning with Rebalanced Dataset (FedReb). It addresses the challenges brought by data heterogeneity in federated learning, overcomes the lack of a generalized global model in personalized federated learning, and demonstrates higher scalability and robustness compared to the reference FedROD algorithm.

Federated Learning with Rebalanced Dataset (FedReb), as a personalized learning solution implemented through data decoupling to address performance degradation caused by data heterogeneity in federated learning, has been validated on image data. It shows performance in global model accuracy and average accuracy comparable to FedROD, a newly proposed algorithm specifically designed to optimize global model accuracy in personalized federated learning. Additionally, compared to the main reference algorithm FedPer, it surpasses in both aspects.

Furthermore, FedReb possesses the scalability and robustness that FedROD lacks, enabling better performance in scenarios with more users and more imbalanced user data distributions. Using MQTT as the communication protocol, its feasibility for deployment on actual devices has been validated. It is believed that FedReb can solve the data heterogeneity problem in federated learning, overcome the lack of a well-performing generalized global model in personalized federated learning, and has the capability to be deployed in real-world applications.

7.2 Future Work

In Chapter 6, the implementation section, section 6.1, it was mentioned that the algorithm was validated on a testbed composed of Raspberry Pi and personal computers using MQTT, capable of completing training. However, due to limitations in the performance of the experimental equipment, only a small number of clients running FedReb with a simple model were simulated. Additionally, the code for the disconnection and reconnection mechanism implemented in the testbed was not fully tested.

In the future, tests could be conducted using devices with higher computational capabilities, such as the NVIDIA Jetson Nano and other products designed for machine learning. This would allow for the testing and validation of the reconnection mechanism in the code, leading to improvements that would enable practical deployment of the algorithm.

Regarding the algorithm itself, although the experimental section tested the generation of rebalanced datasets using different threshold values t, the effects of using other values were not verified. Future research could focus on how to generate a threshold t that allows the algorithm to achieve good results with a smaller rebalanced dataset or how to generate a better rebalanced dataset that can improve global model accuracy.

Similarly, the algorithm was designed from a training architecture perspective. In the machine learning process, there are many aspects that can be modified to enhance algorithm performance, such as the loss function, optimizer, and model. The algorithm still uses basic components from federated learning and machine learning algorithms, such as Cross Entropy loss and SGD optimizer. Investigating whether other loss functions and optimizers could improve the algorithm's performance is another potential research direction.

Bibliography

[1] Federated optimization in heterogeneous networks. https://github.com/litian96/FedProx, 2021. Last accessed: Nov. 22, 2023.

[2] Exploiting shared representations for personalized federated learning (icml 2021). https://github.com/lgcollins/FedRep, 2022. Last accessed: Nov. 22, 2023.

[3] Federated learning based on dynamic regularization. https://github.com/alpemreacar/FedDyn, 2022. Last accessed: Nov. 22, 2023.

[4] gdown. https://github.com/wkentaro/gdown, 2023. Last accessed: Nov. 21, 2023.

[5] Github. https://www.github.com/, 2023. Last accessed: Nov. 21, 2023.

[6] Mtputty. https://ttyplus.com/multi-tabbed-putty/, 2023. Last accessed: Nov. 21, 2023.

[7] Personalized federated learning platform. https://github.com/TsingZ0/PFL-Non-IID, 2023. Last accessed: Nov. 22, 2023.

[8] Raspberry pi. https://www.raspberrypi.org/, 2023. Last accessed: Nov. 21, 2023.

[9] Realvnc. https://www.realvnc.com/, 2023. Last accessed: Nov. 21, 2023.

[10] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. *arXiv preprint arXiv:2111.04263*, 2021.

[11] Vinod Nair Alex Krizhevsky and Geoffrey Hinton. The cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html, 2009. Last accessed: Nov. 18, 2023.

[12] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers, 2019.

[13] Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements, 2019.

[14] Cedalo. Eclipse mosquitto. https://mosquitto.org/, 2023. Last accessed: Nov. 21, 2023.

[15] Hong-You Chen and Wei-Lun Chao. On bridging generic and personalized federated learning for image classification, 2022.

[16] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Exploiting shared representations for personalized federated learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2089–2099. PMLR, 18–24 Jul 2021.

[17] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 113–123, 2019.

[18] Artur Back de Luca, Guojun Zhang, Xi Chen, and Yaoliang Yu. Mitigating data heterogeneity in federated learning with data augmentation. *arXiv preprint arXiv:2206.09979*, 2022.

[19] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[20] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3557–3568. Curran Associates, Inc., 2020.

[21] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008.

[22] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[23] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pages 5132–5143. PMLR, 2020.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, page 179. Citeseer, 1997.

[26] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Survey of personalization techniques for federated learning. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 794–797. IEEE, 2020.

[27] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021.

[28] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[29] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[30] OASIS Standard. Mqtt version 3.1.1 plus errata 01. `https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`, 2015. Last accessed: Nov. 18, 2023.

[31] Jaehoon Oh, Sangmook Kim, and Se-Young Yun. Fedbabu: Towards enhanced representation for federated image classification, 2022.

[32] Deval Shah. The essential guide to data augmentation in deep learning. `https://www.v7labs.com/blog/data-augmentation-guide`, 2022. Last accessed: Nov. 18, 2023.

[33] Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–17, 2022.

[34] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1698–1707, 2020.

[35] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[36] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[37] Han Xiao, Kashif Rasul, and Roland Vollgraf. fashion_mnist. `https://www.tensorflow.org/datasets/catalog/fashion_mnist`, 2017. Last accessed: Nov. 21, 2023.

[38] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International conference on machine learning*, pages 7252–7261. PMLR, 2019.

[39] Jianqing Zhang, Yang Hua, Hao Wang, Tao Song, Zhengui Xue, Ruhui Ma, Jian Cao, and Haibing Guan. Gpfl: Simultaneously learning global and personalized feature information for personalized federated learning, 2023.

[40] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 2021.

Appendix A

Application Used on Testbed

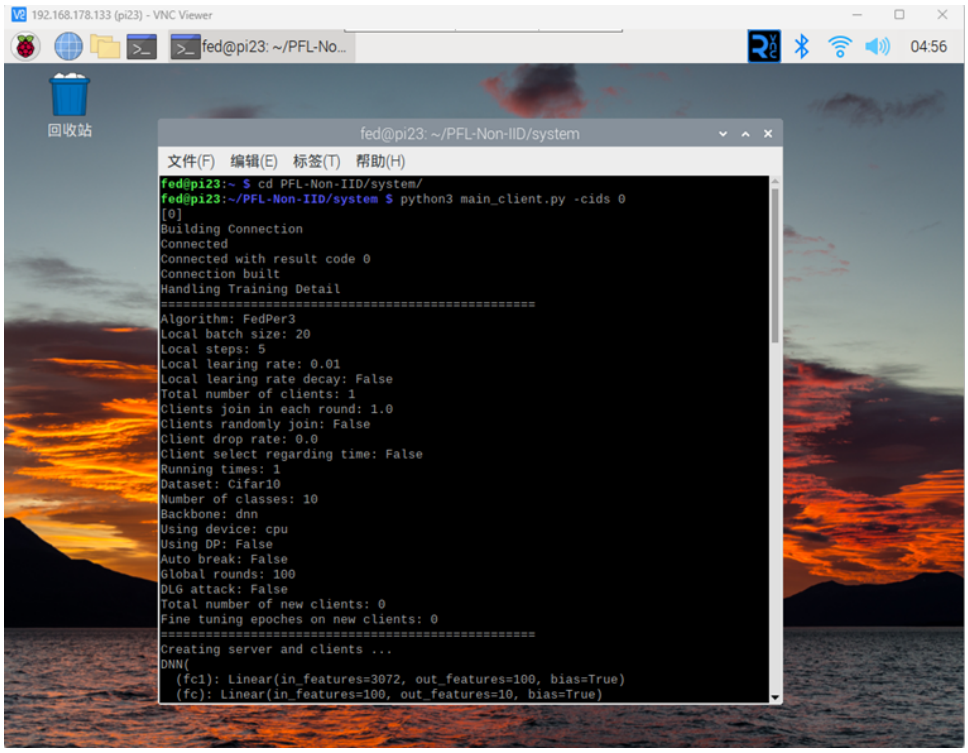


Figure A.1: **VNC Viewer[9] running on a personal computer, connecting to the Raspberry Pi desktop. This setup allows users to interact with the Raspberry Pi desktop without the need for a screen or other external devices.**

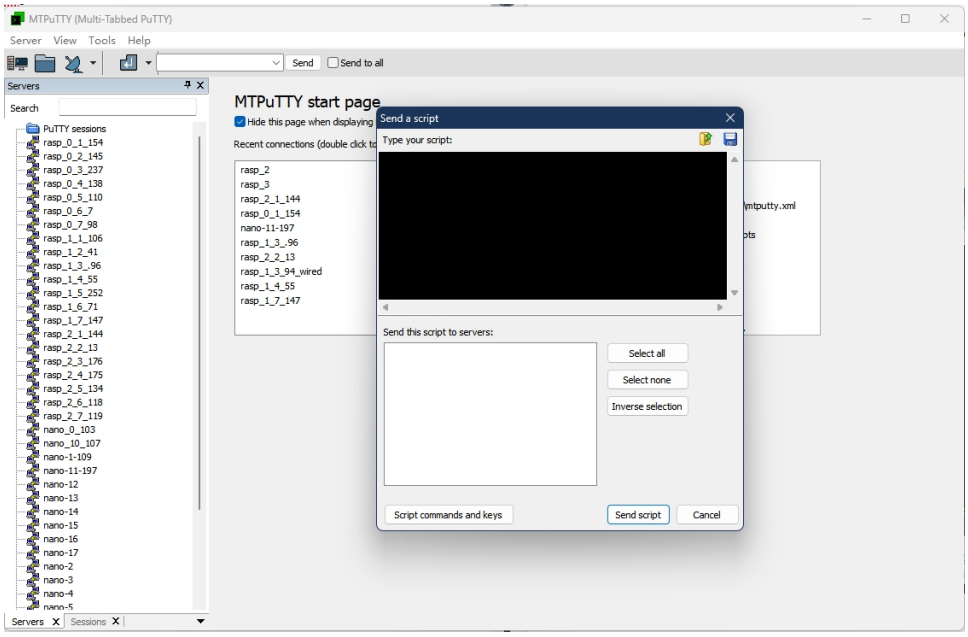


Figure A.2: **MTPuTTY running on a personal computer. User can send script commands to all the SSH clients it connects all at once.**

Appendix B

Source Code

Source Code can be referred from Github Link `https://github.com/TUDelftFedAvg2022/PFL-Non-IID`. The repository may still be private,