Università degli Studi di Padova

An Empirical Study of Decentralized Fine-tuning for Machine Learning Models

Department:	Mathematics
Major:	Computer Science
Name:	Yasser Nabil
Supervisor:	Thomas Marchioro

2024

Abstract

In a world where data collected by edge devices is of high interest, gathering it at a central point has become more challenging due to current data protection regulations.

Decentralized paradigms, such as federated and gossip learning, have gained popularity as a solution to train models while avoiding to share raw data. In decentralized machine learning, edge devices train a local version of the model on their own private data. Moreover, local models are then iteratively shared and merged to produce an aggregated model. Federated learning has been proposed as an initial solution, which relies on a central server for the aggregation. In gossip learning, on the other hand, the model sharing and aggregation is performed in a peer-to-peer fashion, giving peers more control over which peers to collaborate with.

Previous studies tested feasability and limits of decentralized machine learning training models from scratch, which requires access to a large pool of data. In this thesis, we analyze a different aspect of decentralized training, exploring the case of decentralized fine-tuning of machine learning models with a limited amount of available data. Our research involves extensive experimentation with a number of model architectures and datasets, considering different data distributions and imbalances to simulate a real-life setup, demonstrating its applicability across different scenarios.

Keywords: Decentralized Machine Learning, Gossip Learning, Federated Learning, Fine-tuning, Model averaging.

Acknowledgement

I want to express my gratitude to Dr. Thomas Marchioro, my thesis advisor, for always being there for me and pointing me in the correct direction whenever I needed it. Without his competent assistance and encouragement, my thesis would not have been doable. I would also like to thank Padua University for this opportunity given that helped me grow a lot and learn many important concepts and values.

Abbreviations and Acronyms

Abbreviation	**Meaning**
FL	Federated learning
ML	Machine learning
DML	Decentralized Machine learning
AI	Artificial Intelligence
SGD	Stochastic Gradient Descent
IoT	Internet of Things
Non-IID	Non-Independent and Identically Distributed
TN	True negative
TP	True positive
FN	False negative
FP	False positive

Notation Table

Notation	**Meaning**
η	The learning rate
x	Feature vector
y	Class label
n	Number of training examples
m	Number of target classes
N	Number of peers
w	Model parameters
$f_w(x)$	Model
t	Number of iteration
$\mathcal{C}$	Set of classes
$\mathcal{N}$	Set of N peers
D	Dataset
B	Batch size
$J(w)$	Objective function
$\ell(f_w(x), y)$	Loss function
$L(x, y, w)$	Per-example loss function

Contents

Chapter 1

Introduction

Edge devices are well known for producing large volumes of data which is specially needed for machine learning (ML) applications, since they require enormous amounts to achieve high accuracy. The standard approach in ML applications is to collect data from distributed data sources, then convey it to a central node where they continue learning and making decisions. However, collecting data at a central point has become a main challenge in the past years due to current regulations over data privacy [7], along with the overall general awareness of the public on concerns related to data handling.

Decentralized machine learning (DML) paradigms, such as federated and gossip learning, have gained popularity as a solution to avoid sharing raw data while training models. Federated learning (FL), originally proposed by Google [18], has been recognized for its innovative approach to training machine learning models across decentralized networks, allowing data to remain on the local device. It only shares model updates to the server, hence mitigating privacy risks for the peers.

FL relies on a central aggregation node for processing data. This central node or server is responsible for merging the model updates from all devices and distribute the updated global model back to them. While this approach simplifies the coordination of the learning process, it introduces potential scalability challenges. As the number of devices increases, the central node starts limiting the system's ability to scale effectively. Additionally, the server represents a possible point of failure: If the central node is compromised, it could potentially impact the entire network.

As for gossip learning, which is truly decentralized since it uses a peer to peer fashion of aggregating the data instead of relying on a central server, it represents an innovative approach that addresses critical challenges inherent in traditional machine learning techniques [22] [12] [11]. This decentralized approach offers several significant benefits. Similarly to FL, the data never leaves its originating device having devices only share updates. Scalability also represents an important benefit since gossip learning does not rely on a central server.

While gossip learning shows great potential, it also presents some challenges. The random nature of the communication between devices can lead to slower convergence times compared to centralized methods, ensuring that all devices have updated models also presents itself as a great challenge due to the asynchronous nature of the updates.

The primary objective of this research is to explore the potential of decentralized machine learning in different settings not previously explored. Many prior works have studied the applicability and limitations of decentralized machine learning. However, most of these works focused on settings where data providers contribute with a large amount of data, which allows to train complex machine learning models from scratch. On the other hand, a more common case is that the node has limited data and access to pretrained foundation models, which can be fine-tuned for specific tasks. This is a frequent setting in several computer vision and natural language processing applications [24][29]. Our research will focus on observing how fine-tuning could impact the overall performance of decentralized machine learning paradigms, with a clear interest on federated and gossip learning.

To capture various possible settings, we conduct a series of experiments involving the variation of the number of models, datasets, with different distributions and imbalances in data. By doing so, we aim to provide a robust assessment of DML capabilities and limitations.The similarities and differences in their performance provide valuable insights into the strengths and weaknesses of that approach, and we hope to encourage further exploration and adoption of decentralized machine learning methods.

The remainder of this thesis is structured as follows. Chapter 2 covers essential concepts and background information necessary for understanding this research, including the fundamentals of deep learning, decentralized machine learning, and network science. Chapter 3 reviews related work in the field of decentralized machine learning, emphasizing existing research on unbalanced partition sizes, non-IID data, and various averaging techniques. Chapter 4 details the experimental setup used to assess the performance of decentralized fine-tuning across different scenarios.

Chapter 2

Background

In this chapter, we will explain necessary information for understanding the context of this research. This includes an overview of the basics of machine learning, along with in-depth explanation of federated and gossip learning.

2.1 Machine Learning Fundamentals

Machine learning is an important field of artificial intelligence (AI) where we explore how computer systems can improve their perfomance based on experience. ML enables machines to learn from data so that they can make decisions without being explicitly programmed for it. A computer program is said to learn from experience with respect to some class of tasks and performance measure, if its performance at tasks, as measured, improves with experience [19]. This chapter serves as a thorough introduction to the fundamental principles of machine learning, which represents the foundation for the concepts explored in this research.

2.1.1 Goal of Machine Learning

Machine learning involves machines grasping patterns and making decisions without being explicitly programmed for each task. Instead of providing specific instructions, like traditional programming, machine learning algorithms are given data to learn from.

ML tasks usually involve specifying how the system should handle examples. An example consists of features that have been assessed from an object or event that the machine learning system aims to analyze. These examples are commonly represented as vectors x in $\mathbb{R}^n$, where each entry x_i of the vector corresponds to a distinct feature.

Numerous types of tasks can be addressed using machine learning. Among the most prevalent machine learning tasks are classification and regression.

In this research, our focus will be on the classification task as it represents our goal. This task type involves the computer program determining which of c categories some input belongs to. To accomplish this task, the learning algorithm is typically tasked with generating a function $f : \mathbb{R}^n \to \{1, ..., k\}$. When $y = f(x)$, the model assigns an input described by the vector x to a category identified by the numeric code y.

An example of a classification task is object recognition, where the input is an image (represented as a set of pixel brightness values), and the output is a numerical code that identifies the object depicted in the image.

We work with a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ containing n examples. Each example (x, y) comprises a feature vector x in n-dimensional space and its corresponding class label y, which is a categorical designation of a data point to indicate its belonging to a certain class, from the set $\mathcal{C}$.

The primary goal is to determine the parameters w of a function $f_w : \mathbb{R}^n \to \mathbb{C}$ that effectively classifies examples in D and generalizes well beyond it, achieved by minimizing an objective function $J(w)$.

$$w^* = \arg\min_w J(w) = \arg\min_w \left(\frac{1}{n} \sum_{j=1}^{n} \ell(f_w(x_j), y_j) + \frac{\lambda}{2} \|w\|^2 \right) \tag{2.1}$$

The regularization term $\|w\|^2$ and the coefficient λ help in controlling model complexity and preventing overfitting. Next sections will provide a more comprehensive explanation of how this process works.

The loss function $\ell()$ quantifies the prediction error and, in the context of classification tasks, it represents the cross-entropy loss. The objective in classification is to predict the label y for a given input x. To achieve this, the model is designed to output a value for each class. If there are m classes, the model's output for an input x is $f_w(x) = [s_1, \ldots, s_m]$, where $s_k \in \mathbb{R}$ is the score for the k-th class. These scores are converted into probabilities $[q_1, \ldots, q_m]$ using the softmax activation function:

$$q_k = \frac{e^{s_k}}{\sum_{l=1}^{m} e^{s_l}} \tag{2.2}$$

Each q_k lies between 0 and 1, and the sum of all q_k's equals 1.

The cross-entropy loss is the standard loss function used in classification problems. It calculates the error based on the probability assigned to the true class y. The loss function is defined as:

$$\ell(f_w(x), y) = -\log(q_y) \tag{2.3}$$

This loss function yields higher values (indicating larger errors) when the probability q_y is low. Conversely, the loss approaches 0 as q_y approaches 1.

2.1.2 The Classification Task

When data is not complete, classifying it becomes tougher for computer programs. In typical classification tasks, algorithms create one function linking input vectors to categorical outputs. Yet, if some inputs are absent, the algorithm must learn multiple functions, each handling x with different missing subsets.

This distinction between complete and incomplete data is particularly relevant when considering different learning paradigms. Machine learning algorithms can be broadly categorized as supervised or unsupervised based on the type of experience they are allowed to have during the learning process.

In supervised learning, where algorithms are trained on labeled data to predict outputs accurately, handling missing data is crucial to ensure effective model training. The goal in this type of learning is to learn a decision rule $f : X \to Y$ using some labeled pairs (x_j, y_j) with $j = 1, ..., n$ and $x_j \in X, y_j \in Y$.

Regression tasks, which involve predicting numerical values based on inputs, also face challenges with incomplete data. The learning algorithm is tasked with producing a function $f : \mathbb{R}^n \to \mathbb{R}$. While similar to classification, the key distinction lies in the format of the output. Algorithms focus on estimating relationships between variables to predict continuous outcomes, making the handling of missing data equally important in regression scenarios.

On the other hand, unsupervised learning involves discovering patterns and structures in data without explicit labels. Usually, this type of learning leverages similarities and differences within the data. In this type of learning, the algorithm must infer relationships and patterns solely from the input data, making the handling of missing data equally important. The absence of labels necessitates robust methods for dealing with incomplete data, as the algorithm relies solely on the inherent structure of the data for learning. In deep learning, which is the most prominent subfield of machine learning and represents an umbrella term for all machine learning applications that utilize neural network models, the goal often involves learning the entire probability distribution that generated a dataset, either explicitly (in density estimation) or implicitly (tasks like denoising).

Classification comes in two main types: binary classification and multi-class classification. In binary classification, the machine decides between only two options, while in multi-class classification, there are more than two choices, which adds complexity to the task.

Performance evaluation in machine learning is crucial for assessing the effectiveness of models. For tasks like binary classification, where there are only two possible outcomes, and multi-class classification, where there are more than two classes, accuracy is a common measure of performance. Accuracy represents the proportion of examples for which the model produces the correct output.

In addition to accuracy, various metrics evaluate the performance of machine learning models in both binary and multi-class classification tasks.

Binary Classification

For binary classification, these metrics include accuracy, precision, recall, specificity, balanced accuracy, and F1 score.

- Accuracy represents the probability of correct guesses given a random sample from the test dataset, limitations to this measure is that it may not represent a good performance measure when the test dataset is unbalanced;

$$Accuracy = \frac{n_{correct}}{n_{total}} \tag{2.4}$$

- Precision measures the proportion of correctly predicted positive cases out of all cases predicted as positive;

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

  where true positive (TP) refers to the number of instances where the model correctly predicts the positive class, and false positive (FP) as the number of instances where the model incorrectly predicts the positive class when it is actually negative.

- Recall quantifies the model's ability to identify all relevant instances among all actual positive cases;

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

  where false negative (FN) refers to the number of instances where the model incorrectly predicts the negative class when it is actually positive.

- Specificity measures the proportion of correctly identified negative cases among all actual negative cases;

$$Specificity = \frac{TN}{TN + FP} \tag{2.7}$$

  where true negative (TN) refers to the number of instances where the model correctly predicts the negative class when it is actually negative.

- Balanced accuracy calculates the average of sensitivity and specificity to provide a comprehensive measure of model performance;

$$Balanced\ accuracy = \frac{1}{2}Recall + \frac{1}{2}Specificity \tag{2.8}$$

- The F1 score is a harmonic mean of precision and recall, it offers a balanced assessment of a model's performance by considering both false positives and false negatives;

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{2.9}$$

Multi-class Classification

Similarly, for multi-class classification, accuracy, balanced accuracy, and F1 score are common metrics used to evaluate model performance. In this case, balanced accuracy is the average of Recalls for all classes. For instance, in a dataset with 3 classes we get

$$Balanced\ accuracy = \frac{1}{N}\sum_{c=1}^{N} Recall_c \quad (2.10)$$

Where $Recall_c$ denotes the recall computed for class. As for F1 score, it is the average F1 score for all classes.

$$F1_{avg} = \frac{1}{N}\sum_{c=1}^{N} F1_c \quad (2.11)$$

Where $F1_c$ denotes the F1 score computed for each class c, and N is the total number of classes.

2.1.3 Generalization and Model Complexity

The primary challenge in machine learning lies in ensuring that our algorithm performs effectively on new, previously unseen inputs. The ability to perform well on previously unseen data is called "generalization".

A machine learning model is trained on a training dataset and, subsequently, its generalization capability is assessed on a test dataset. During training, the model is tuned to minimize a certain error measure on the training data. This is done by treating the error minimization as an optimization problem.

However, even if the optimal value is found for the training dataset, the same performance is not guaranteed for data which was not included in the dataset. The expected value of the error on unseen data, called generalization error, is estimated by measuring the error on the test dataset.

Machine learning algorithms tend to achieve optimal performance when their capacity aligns with the true complexity of the task they are intended to address and the volume of training data available. Controlling the capacity of a learning algorithm can be achieved by specifying its hypothesis space, which denotes the set of functions that the learning algorithm can choose from as potential solutions.

Models with inadequate capacity may struggle to tackle complex tasks effectively. On the other hand, models with high capacity can address complex tasks proficiently; however, if their capacity exceeds what is necessary for the current task, they may overfit the training data.

Overfitting occurs when a model learns the training data too well, capturing noise or random fluctuations, which leads to poor generalization on unseen data. Inversely, underfitting happens when the model is too simple to capture the underlying structure of the data, which results in poor performance on both the training and unseen data.

Balancing model complexity is of great importance, as it influences the trade-off between underfitting and overfitting. Hence the need for techniques such as controlling the hypothesis space and quantifying model capacity using the VC dimension. We can influence a learning algorithm with a preference for one solution over another within its hypothesis space. This implies that both functions are admissible, but one is favored. The less preferred solution will be selected only if it significantly outperforms the preferred solution in fitting the training data, this is the regularization technique.

2.1.4 Hyperparameters and Validation Sets

Most machine learning algorithms are characterized by hyperparameters, which are settings used to govern the behavior of the algorithm. In some cases, a setting is designated as a hyperparameter because it is challenging for the learning algorithm to optimize. More commonly, a setting becomes a hyperparameter because it is not suitable to learn that parameter from the training set. If these hyperparameters were learned from the training set, they would invariably select the maximum possible model capacity, leading to overfitting.

To address this challenge, we use a validation set of examples that the training algorithm does not observe. A held-out test set consists of examples from the same distribution as the training set. It is used to estimate the generalization error of a learner once the learning process concludes. It is crucial that the test examples are not used in any manner to make decisions about the model, including its hyperparameters. Hence, no example from the test set can be included in the validation set. Consequently, we always construct the validation set from the training data.

In fact, we partition the training data into two separate subsets. One subset is employed for learning the parameters, while the other serves as our validation set, used to estimate the generalization error during or after training. This enables us to update the hyperparameters accordingly. Usually, around 80 % of the training data is allocated for training, while the remaining 20 % is designated for validation.

Dividing the dataset into a fixed training set and a fixed test set can pose challenges, particularly if the test set ends up being small. A small test set leads to statistical uncertainty around the estimated average test error, making it difficult to confidently assert that algorithm A performs better than algorithm B on the given task.

While this is not a serious issue when the dataset contains hundreds of thousands of examples or more, it becomes problematic with smaller datasets. In such cases, alternative procedures allow one to utilize all examples in estimating the mean test error, at the cost of increased computational overhead. These procedures are based on the concept of repeating the training and testing computations on different randomly chosen subsets or splits of the original dataset.

One common approach is k-fold cross-validation, where the dataset is partitioned into k non-overlapping subsets. The test error is then estimated by averaging the test error across k trials. In each trial, one of the k subsets serves as the test set, while the remaining data is used for training.

Hence, hyperparameters and validation sets play a pivotal role in fine-tuning machine learning models to achieve optimal performance. Since, we need to carefully select hyperparameters and validate model performances on separate validation sets to ensure of a good generalization. This iterative process of hyperparameter tuning is closely intertwined with optimization techniques such as stochastic gradient descent, since it serves as an optimization algorithm, enabling models to efficiently navigate parameter space and converge towards optimal solutions.

2.1.5 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an algorithm playing an important role in the field of deep learning. A common issue in machine learning is the necessity for large training sets to achieve good generalization, yet such sets are computationally expensive. Typically, the cost function used by a machine learning algorithm decomposes as a sum over training examples of a per-example loss function. For instance, the negative conditional log-likelihood of the training data can be expressed as:

$$J(w) = \mathbb{E}_{x,y\sim\hat{p}_{data}} L(x, y, w) = \frac{1}{m}\sum_{j=1}^{m} L(x^{(j)}, y^{(j)}, w) \tag{2.12}$$

$J(w)$ denotes the cost function parameterized by w, $\hat{p}_{data}$ represents the empirical distribution over training data, $L(x, y, w) = \ell(f_w(x), y)$ denotes the per-example loss function, and m is the number of training examples. The cost function is computed as the average loss over all training examples.

For such additive cost functions, gradient descent necessitates computing the gradient of the cost function with respect to the parameters w, denoted as $\nabla_w J(w)$. This gradient is computed as the average of the gradients of the per-example loss functions over all training examples:

$$\nabla_w J(w) = \frac{1}{m} \sum_{j=1}^{m} \nabla_w L(x^{(j)}, y^{(j)}, w) \tag{2.13}$$

$\nabla_w L(x^{(j)}, y^{(j)}, w)$ represents the gradient of the per-example loss function with respect to the parameters w.

The computational cost of computing the gradient of the cost function with respect to the parameters w in this manner is $O(m)$. However, as the size of the training set grows to billions of examples, the time required to take a single gradient step becomes prohibitively long. This limitation poses a significant challenge in practical applications of machine learning, particularly in deep learning where large-scale datasets are common.

The key insight of stochastic gradient descent is that the gradient can be approximated as an expectation, which can be estimated using a small set of samples. Specifically, in each step of the algorithm, a mini-batch of examples $\mathcal{B} = \{x^{(1)}, \ldots, x^{(B)}\}$ is sampled uniformly from the training set. The mini-batch size B is typically chosen to be a relatively small number of examples, ranging from one to a few hundred.

B is usually held fixed as the training set size m grows. This allows for training on a dataset with billions of examples using updates computed on only a hundred examples.

$$g = \frac{1}{B} \nabla_w \sum_{j=1}^{B} L(x^{(j)}, y^{(j)}, w) \tag{2.14}$$

g represents the estimate of the gradient, computed as the average of the gradients of the per-example loss function over the mini-batch B, where B is the mini-batch size.

Using examples from the mini-batch B, the stochastic gradient descent algorithm then updates the parameters as follows: $w \leftarrow w - \eta g$, where η is the learning rate.

Hence, the SGD emerges as a prevalent approach for determining w^* Equation (2.1). Initially, we set an initial weight vector w_0. The iterative process involves updating w through successive iterations or epochs, where each epoch represents a complete pass through the entire dataset. Each iteration typically involves selecting a random example (x_j, y_j) from the dataset. The learning rate η_t governs the size of the step taken in each update.

$$w_{t+1} = w_t - \eta_t \left(\lambda w_t + \frac{\nabla_w \ell(f_w(x_j), y_j)}{w} \right) \tag{2.15}$$

This equation represents the update rule used in SGD, where η_t represents the learning rate at iteration t. In practice, data is often divided into batches to improve computational efficiency during training. After training, the model's performance is evaluated on a separate test dataset to assess its generalization ability.

Through exploring the goal of machine learning, various performance measures, learning paradigms involving supervised and unsupervised learning, as well as the critical aspects of generalization, regularization, hyperparameters, validation sets, and stochastic gradient descent, we have established a solid groundwork. We are now well prepared to delve deeper into more complicated topics.

2.2 Decentralized Machine Learning

In the field of machine learning, conventional methods typically depend on centralized data storage, where large volumes of data are gathered, processed, and examined in a single location. This centralized model presents notable obstacles, such as risks to privacy, inefficient communication, and scalability limitations. In answer to these challenges, decentralized machine learning has presented itself as a valid alternative, providing a distributed framework where data stays localized, and computations are executed cooperatively across different nodes.

Decentralized machine learning (DML) refers to the process of training machine learning models across a network of distributed devices, where each device holds local data and contributes to the model training process without sharing raw data externally. The significance of DML lies in its ability to harness the collective intelligence of different datasets while respecting more data privacy and security concerns, by manipulating data directly from edge devices, such as smartphones, IoT devices, and edge servers. In the next sections, we will introduce two of the main DML paradigms we focus on in this research.

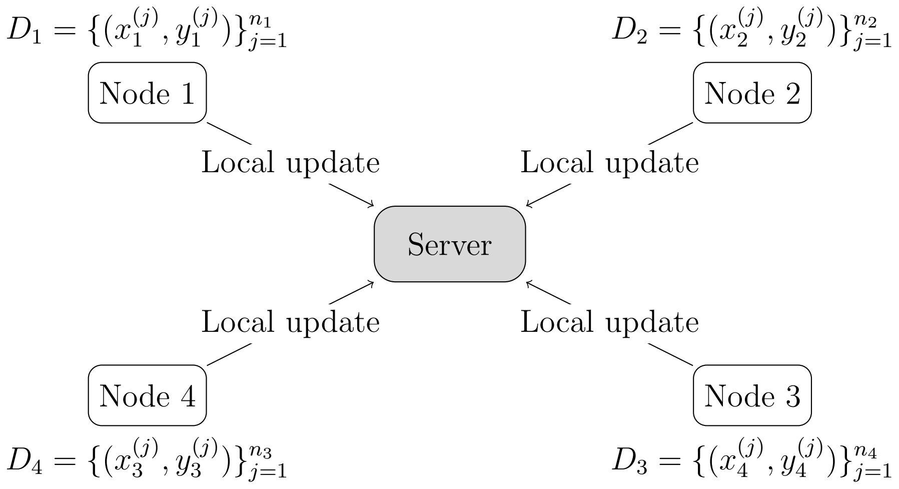
2.2.1 Federated Learning

Driven by concerns regarding privacy among data owners, the concept of federated learning (FL) is introduced by Google [18]. FL enables users to collectively train a shared model while retaining their personal data on their respective devices, hence improving their privacy. Consequently, FL emerges as a facilitating technology for training machine learning models within mobile edge networks.

Algorithm 1 Local training function [18].

```
1: Input: $f_w$ is the model weights; $D_i$ is the local dataset; $B$ is the mini-
   batch size; $E$ is the total number of epochs; $\eta$ is the learning rate
2: Return: Updated model weights $w$
3:
4: function LocalTraining($f_w$):
5:     $\mathcal{B}_1, \ldots, \mathcal{B}_K \leftarrow$ Split $D_i$ to minibatches of size $B$
6:     for each epoch $e$ from 1 to $E$ do
7:         for k = 1, ..., K do
8:             Compute g on $\mathcal{B}_k$ according to (2.14)
9:             $w \leftarrow w - \eta g$.
10:        end for
11:    end for
12: end function
```

A federated learning system consists of two primary entities: the data owners, also called peers or nodes, and the model owner, often referred to as the central server or aggregator. Let $\mathcal{N} = \{1, \ldots, N\}$ represent the set of N peers, each possessing a private dataset $D_i \in \mathcal{N}$. Each peer $i, \ldots, N$ utilizes its dataset D_i to train a local model (Algorithm 1) and transmits only the local model parameters w_i to the FL server. Subsequently, all collected local models are aggregated into a global model with weight w_G (Figure 2.1). This approach differs from traditional centralized training, where data from all sources are aggregated before centrally training a model.



$D_1 = \{(x_1^{(j)}, y_1^{(j)})\}_{j=1}^{n_1}$
Node 1
$D_2 = \{(x_2^{(j)}, y_2^{(j)})\}_{j=1}^{n_2}$
Node 2
Local update
Local update
Server
Local update
Local update
Node 4
Node 3
$D_4 = \{(x_3^{(j)}, y_3^{(j)})\}_{j=1}^{n_3}$
$D_3 = \{(x_4^{(j)}, y_4^{(j)})\}_{j=1}^{n_4}$

Figure 2.1: Federated learning network example.

Algorithm 2 Federated averaging algorithm [18].

```
1: Input: $\mathcal{N}$ is the set of peers; $T$ is the total number of iterations; $w_G^0$ is
   the initial global model weights; LocalTraining is the local training
   function
2: Return: Final global model weights $w_G$
3:
4: Initialize $w_G$
5: for each iteration $t$ from 1 to $T$ do
6:     Randomly choose a subset $S_t$ of $x$ peers from $\mathcal{N}$
7:     for each peer node $i \in S_t$ parallelly do
8:         $w_i \leftarrow \text{LocalTraining}(f_{w_G})$
9:     end for
10:    $w_G \leftarrow \frac{1}{\sum_{i \in N} D_i} \sum_{i=1}^{N} D_i w_i$
11: end for
```

FL peers collaborate to train a machine learning model requested by an aggregate server. The server initiates the training task by determining the goal and its associated data requirements. The server then specifies the hyperparameters of the global model and the training process, such as the learning rate, batch size, and number of epochs per training round. The server then shares the initialized global model w_G and task details to the devices.

Utilizing the global model w_G each peer individually uses its local data and device to update the local model parameters w_i. The objective for peer i in iteration t is to determine the optimal parameters w_i^* that minimize the loss function $\ell(w_i)$, expressed as:

$$w_i^* = \arg\min_{w_i} J(w_i) \tag{2.16}$$

Following this update, the peer transmits the revised local model parameters to the server (Algorithm 2). The server then updates the global model parameters w_G using the received local models contributed by peers. The server aims to minimize the global loss function $\ell(w_G)$.

While this method simplifies the coordination of the learning process, it presents potential scalability challenges. As the number of devices increases, the server's ability to scale effectively becomes more limited. Hence the need for an alternative: Gossip learning, which is a decentralized communication protocol offering a different approach to address these concerns.

2.2.2 Gossip Learning

Gossip learning represents a decentralized machine learning method that operates without a central control over fully distributed data. In this setup, we presume that the dataset D is distributed horizontally across multiple nodes, where each node i holds its own subset D_i.

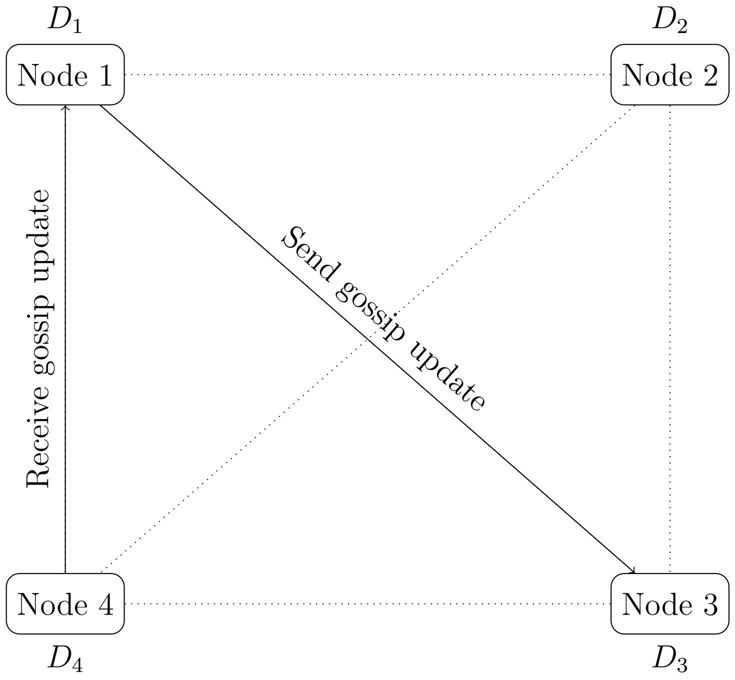


Figure 2.2: Gossip learning network example.

Each node i executes Algorithm 3. Initially, the node initializes its local model (w_i) and its age t_i. A portion of the model parameters is periodically transmitted to another node within the network. Upon receiving such a parameter sample, a node adds it into its own model and then executes a local update step. Although all nodes adhere to the same period $\Delta\tau$, the rounds are not synchronized. Various versions of the algorithm can be developed by employing different implementations of the methods sample, merge, and update. In its simplest form, the sample method transmits the entire model (without sampling), the merge method calculates the average, and the update method performs a mini-batch update based on local data.

Gossip learning proved itself applicable in various domains including distributed sensor networks [23], peer-to-peer networks [20], and Internet of Things (IoT) where data privacy, fault tolerance, and decentralization are of great importance. While gossip learning algorithms can offer benefits, one potential challenge is ensuring that the learning process converges to an accurate model. The convergence rate and the final model accuracy can be influenced by several factors, including the number of devices, the frequency

Algorithm 3 Gossip Learning Algorithm.

```
1: Input: $\Delta\tau$ is the time interval; $t$ is the communication round; $w$ is the model weights LocalTraining is the local training function
2: Return: Final parameters $t_i$, $w_i$ after convergence
3:
4: Initialize parameters: $t_i = 0$, $w_i = 0$
5: repeat
6:     Wait for time interval $\Delta\tau$
7:     Select a peer node $i$
8:     Transmit model sample $(t_i, w_i)$ to node $i$
9: until convergence
10:
11: procedure onReceiveModel($t_r, w_r$)
12:     Merge received model $(t_r, w_r)$ into local model $(t_i, w_i)$
13:     $w_i \leftarrow$ LocalTraining($f_{w_i}$)
14:     $t_i \leftarrow \max(t_i, t_r) + 1$
15: end procedure
```

in which they communicate between them, and the diversity of the data distribution across nodes.

Each iteration of the gossip learning algorithm involves nodes sharing their model parameters to other nodes. In a large-scale network, this could lead to an important communication cost. Techniques to reduce this cost could be using a better sampling method to select specific parameters to share. The performance of gossip learning can also depend heavily on how the sample, merge, and update methods are implemented.

2.3 Fine-Tuning

Fine-tuning refers to the process of taking a pre-trained model (model that was trained for a specific task before) and further training it on a specific task or dataset of interest. In the context of ML, this process is applied to large neural networks trained on extensive datasets for tasks like image classification or natural language processing. It enables leveraging existing knowledge for new tasks with minimal data and computational resources. It is closely linked to transfer learning, where knowledge from one task informs another. Fine-tuning adjusts the pre-trained model to better suit the nuances of the target task, making it a common strategy in transfer learning.

Fine-tuning offers several advantages over training a model from scratch for a new task. Pre-trained models have learned rich feature representations from vast amounts of data, which can significantly speed up the training process and require less labeled data for the target task. Additionally, fine-tuning often leads to better generalization performance, as the pre-trained model has already captured generic patterns from the source task, which can be beneficial for related tasks.

The typical workflow for fine-tuning involves four steps as described in [27] and shown in Algorithm 4. First, we train a neural network model, known as the source model w_s, on a specific dataset. Then, we develop a new neural network model, referred to as the target model w_t, replicating all design aspects and parameters from the source model, except for the output layer. The parameters of the source model encapsulate knowledge from the source dataset D_s, assumed to be transferable to the target dataset D_t. The output layer from the source model, closely tied to source dataset labels, is omitted in the target model. We introduce an output layer to the target model, tailored to the number of categories in the target dataset. Initialize the parameters of this layer randomly. Finally, we train the target model on the target dataset. The output layer undergoes training from scratch, whereas the parameters of all other layers are fine-tuned based on those of the source model.

Algorithm 4 Fine-tuning example.

```
1: **Input:** $f_w$ is the pretrained model; $D_t$ is the target dataset
2: **Return:** Fine-tuned target model weights $w_t$
3:
4: $m \leftarrow$ number of unique labels in $D_t$
5: $f'_w \leftarrow f_w$ with last layer replaced by a fully connected layer with $m$ outputs
6: $w_t \leftarrow$ LocalTraining($f'_w$)
7:
8: **return** $w_t$
```

To optimize the fine-tuning procedure and boost the effectiveness of the fine-tuned model we use various techniques. These include adjusting learning rate schedules, implementing data augmentation strategies, and making task-specific modifications to the model architecture.

While fine-tuning offers advantages, it also presents various factors to consider. A potential challenge is overfitting to the target task, especially when dealing with a small dataset or one that differs significantly from the original task. Regularization techniques such as dropout, which randomly deactivates neurons during training, and weight decay, which penalizes large parameter values, are often employed to mitigate overfitting during fine-tuning. Additionally, selecting an appropriate pre-trained model and fine-tuning strategy requires careful consideration of factors such as task similarity, dataset size, and computational resources.

2.4 Fundamentals of Network Science

Network science studies the structure and dynamics of networks. This field of research seeks to understand complex systems by analyzing the interaction of individual components within a network. Network science studies general properties of networks that can be applied to many different fields, including social networks, biological systems, computer networks, and more. In the context of decentralized machine learning, network science can be used to analyze the convergence of a model in different types of networks.

In this section, we introduce different properties of networks and discuss how these affect the convergence of a model.

2.4.1 Degree Distribution

One of the most important properties for nodes in a network is their degree, which is the number of other nodes with whom they are connected. Nodes with higher degree often play a pivotal role in networks, especially when message propagation is involved, as in the case of decentralized machine learning.

In general, the degree distribution in a network, i.e., the probability of a node having a certain degree, determines how interconnected a network is. If we denote with k the degree of a node, the degree distribution of a network can be fully characterized by specifying the probability $p_K(k)$ for each possible value of k.

A possible distinction between different network types is based on their degree distribution. Two widely studied degree distributions are the Poisson distribution and the power-law distribution.

Poisson distribution and random networks A Poisson distribution has the following expression:

$$p_K(k) = \frac{\lambda^k}{k!} e^{-\lambda} \tag{2.17}$$

where $\lambda > 0$ is a parameter that characterizes it. A network that follows a Poisson distribution of the degrees is also called a *random network*. A network generated by connecting pairs of nodes with constant probability p follows a Poisson distribution, as long as the number of nodes N is large. Generally, random networks lack highly connected nodes, which causes the spread of messages to be more uniform and slower, as no single node can serve as a major conduit for information dissemination.

Power-law distribution and scale-free networks A power-law distribution has the form

$$p_K(k) \propto k^{-\gamma} \tag{2.18}$$

and here the parameter γ, which typically takes values between 2 and 3, characterizes the distribution. Networks that follow a power-law distribution are called *scale-free networks*. This type of distribution is especially important, since several real-world networks are scale-free, including Internet hyperlinks, power grids, social networks, and more. A power-law distribution of the degree implies that most nodes have few connections, while a small number of nodes, known as *hubs*, have a very high degree. Hubs play a central role in the network's connectivity and facilitate rapid information dissemination. When a message reaches a hub, it can quickly spread to many other nodes due to the hub's numerous connections, thus accelerating the propagation process.

2.4.2 Graph Representation and Adjacency Matrix

To uniquely define the topology of a network, which is its graph, you need to specify the set of nodes and the links connecting them. Nodes can be identified by assigning each one a unique number from 1 to N, where N is the total number of nodes. After numbering the nodes, the network topology can be uniquely represented by the adjacency matrix A, where the entries indicate the links between the nodes. The adjacency matrix is an $N \times N$ matrix, whose entries can be either 0 or 1 according to the following criterion:

$$A_{ij} = \begin{cases} 1, & \text{if } i \text{ is connected to } j \\ 0, & \text{otherwise} \end{cases} \tag{2.19}$$

Since we consider only undirected network graphs, it must always be $A_{ij} = A_{ji}$. Additionally, we do not allow loops, meaning that $A_{ii} = 0$ for all $i = 1, \ldots, N$. An example of adjacency matrix for a given graph is reported in Fig. 2.3.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

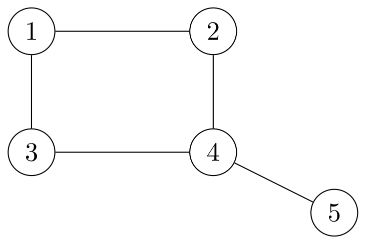


Figure 2.3: Example of adjacency matrix and corresponding graph topology.

2.4.3 Network models

When analyzing network protocols, it is often useful to generate synthetic networks with specific topologies or properties. In this work, we aim to test model convergence in networks with particular degree distributions. To this end, we utilize two well-established algorithms to create synthetic networks: the Erdős-Rényi algorithm, which generates networks with a Poisson degree distribution, and the Barabási-Albert algorithm, which generates networks with a power-law degree distribution.

Erdős-Rényi The Erdős-Rényi algorithm constructs a random network given the number of nodes N and a link probability p. The rationale of the algorithm is quite simple: it initializes a graph of N disconnected nodes, and for each possible node pair (i, j) it creates a link between them with probability $p \in [0, 1]$.

Clearly, the higher the value of p, the more connected is the resulting network. Since each node i is connected to every other node with probability p, the average degree in an Erdős-Rényi network is given by

$$\mathbb{E}[d] = N \cdot p. \tag{2.20}$$

Furthermore, one can observe that the degree distribution is binomial with parameters N and p, which approaches a Poisson distribution with $\lambda = N \cdot p$ for large values of N.

Barabasi-Albert Contrary to the Erdős-Rényi approach, the Barabasi-Albert algorithm starts with a small network and gradually introduces new nodes until the desired number of nodes N is reached. Each new node has a fixed number ν of available links and uses them to connect to existing nodes. The core idea behind this algorithm is preferential attachment: new nodes are more likely to connect to nodes with higher degree. This aims to mimic the dynamics of real-world networks, where "popular" nodes tend to become more and more popular as the network evolves. For each existing node i in the network, the probability of being chosen by a new node is proportional to its degree d_i.

For N large enough, this algorithm generates a network where the degrees follow a power-law distribution with $\gamma \approx 3$. The average degree is simply

$$\mathbb{E}[d] = 2 \cdot \nu \tag{2.21}$$

as every new node introduces 2 new links, which contribute to both their degree and to the degree of the nodes they connect to.

Chapter 3

Related Works

This chapter serves as an introduction to the existing literature on federated and gossip learning, covering the most important works related to these topics. By examining previous researches, this section highlights the contributions of previous researchers but also provides a foundation for this research.

We prioritize gathering various perspectives and conducting a critical evaluation of the strengths and limitations of the existing research. This helps us to establish the groundwork for interpreting and discussing the findings in the next sections.

3.1 Non-IID Data

In decentralized machine learning, the class distribution inside the data between peers significantly affects the performance and efficiency of the resulting models. To verify the effectiveness of proposed solutions in real-world situations, it is essential to test these models with non-IID (Non-Independent and Identically Distributed) data. Non-IID data represents the natural disposition of real-world datasets, where each peer's data may not be evenly distributed. For example, in the case of labeled data (used in classification problem), non-IID data means that different peers might own data with different proportion of the classes. A non-IID distribution may also result in peers having samples from only part of the classes. By assessing models in these conditions, we can gain insights on the robustness and adaptability of our methods. In this section, we will focus on their method for creating non-

IID data, and in the next sections, we will delve deeper into their approaches to addressing the associated challenges.

The paper [21] discusses possible issues arising from peers having different data distribution and proposes a methodology to mitigate these issues. In federated learning, peers are expected to contribute the same type of data drawn from one global distribution. However, data is often collected in different ways from different resources, leading to different marginal distributions among peers compared to the underlying global distribution. The authors' main idea is to adjust the peer distribution closer to the global distribution using sample weights. This adjustment allows the machine learning model to converge faster with higher accuracy.

The authors mimicked non-IID data by equally dividing the data into 100 partitions and introducing varying levels of noise to each peer's data, inspired by the skewness simulation in [15]. The noise was drawn from a Gaussian distribution with a mean of 0 and differing standard deviations. Specifically, for the k-th peer ($k \in [0, 99]$), the noise was added with a variance of $k \times \frac{x}{100}$, where x denotes the added noise variance.

This procedure is one of many ways to emulate a skewed data distribution across peers.

Another approach to addressing this critical problem in the federated learning environment is presented in the paper [18], which originally introduced federated learning. The authors conducted extensive experiments using five different model architectures and four datasets. They tested iterative model averaging, demonstrating that their method significantly reduces communication costs compared to traditional communication rounds. They argue that this approach is effective for both unbalanced and non-IID data.

The authors used a sharding technique, specifically for the MNIST dataset (which is a dataset of labeled digits between 0 and 9), to recreate non-IID data. In contrast to the previous approach, which simulated non-IID data by adding noise to the samples, this method assigns different distributions of labels to different peers. The authors first sorted the data by digit labels, then divided it into 200 shards of size 300, and assigned each of the 100 peers 2 shards. This created a pathological non-IID partition of the data, as most peers only had examples of two digits. This setup allowed them to explore the extent to which their algorithms could handle highly non-IID data distributions.

This approach appealed to us more because we will be sharding our data to mimic the non-IID state, but using 3 shards instead of 2 for a better distribution of the data in our experimental setup.

3.2 Unbalanced Partition Sizes

The distribution of available data to different peers for training can have an important impact on the performance of the models being trained.

Data imbalance is of common occurrence in real-world scenarios, where some peers may have access to a large amount of data, while others may have very little. This difference can lead to significant performance degradation in the global model. Since in decentralized machine learning it relies on training with multiple peers, peers with less data can disproportionately influence the overall model.

This issue is widely addressed in various research papers due to its impact on the performanceof models. This issue is addressed in the paper [3]. The authors acknowledge data heterogeneity across peers in FL settings as a widely acknowledged challenge. They propose a novel method, FedNH, that improves the local models' performance for both personalization and generalization by combining the uniformity and semantics of class prototypes.

Moreover, data imbalance can also affect the convergence speed of the model. In FL for example, peers with more data contribute more to the global model update, which can cause the model to converge towards the data distribution of these peers faster. This can result in a model that is biased towards the data-rich peers, potentially at the expense of performance on data-poor peers.

In the next section, we will present how some researchers were able to tackle the challenges of non-IID data and unbalanced data by implementing new averaging techniques.

3.3 Averaging Techniques

In decentralized machine learning, aggregating parameters from different peers is essential. This section examines various averaging techniques proposed in prior studies to address possible challenges. By assessing these approaches, this section aims to identify their strengths, weaknesses, and implications for decentralized model refinement in machine learning.

In the paper "Communication-Efficient Learning of Deep Networks from Decentralized Data" [18] introduces a method for federated learning based on iterative model averaging, named $FederatedAveraging$ or $FedAvg$.

Algorithm 2 illustrates how the server aggregates model updates from participating clients or peers in a federated learning setting. Initially, the server initializes the global model parameters (w_0). Then, in each communication round ($t = 1, 2, \dots$), a subset of peers (S_t) is randomly chosen. Each selected peer updates its model parameters (w_{t+1}^k) based on its local data and transmits these updates to the server. Once all peers complete their updates, the server computes the weighted average of these parameters to derive the new global model parameters (w_{t+1}).

FederatedAveraging demonstrates an averaging approach that utilizes decentralized computation to aggregate model updates from diverse data sources, addressing challenges arising from data imbalances among peers. Through iterative collaboration, it facilitates the creation of a global model that integrates insights from all participating peers, thereby enhancing the overall performance and resilience of the federated learning system.

The iterative model averaging technique proves particularly effective in scenarios where data is unbalanced and non-IID (non-Independently and Identically Distributed) across different peers. The authors conducted an extensive empirical evaluation using five different model architectures and four datasets. Their results demonstrated that $FedAvg$ trains high-quality models with relatively few communication rounds, as evidenced by its performance across various architectures: a multi-layer perceptron, two different convolutional neural networks, a two-layer character LSTM, and a large-scale word-level LSTM.

The paper [21] starts from the fundamental concept of empirical risk minimization and theoretically derive a solution for adjusting the distribution skewness using sample weights. The main idea is to adjust the weight of each sample in the loss function during training, which effectively adjusts the peer's data distribution closer to the global distribution.

The goal is to approximate the true global distribution $p(x)$ as closely as possible. To achieve this, it is necessary to find the appropriate adjusting weights $\alpha_k(x)$ for the data distribution of the k-th peer $q_k(x)$, as shown in equation (3.1) from [21]:

$$\alpha_k(x)q_k(x) = p(x) \tag{3.1}$$

To determine sample weights, the authors utilize a neural network-based density estimation model, MADE (Masked Autoencoder for Distribution Estimation) [8], to implicitly exchange density information without exposing raw data.

MADE is an autoencoder modification that produces powerful generative models by masking the autoencoder's parameters to respect autoregressive constraints. This ensures that each input is reconstructed only from previous inputs in a specific order. Under these constraints, the autoencoder outputs represent conditional probabilities, whose product yields the full joint probability.

The central server calculates sample weights for each peer using the estimated global data distribution. These weights adjust the peers' local models during federated learning, aligning their local data distributions with the global distribution. This method addresses the challenge of skewed data distribution across peers in federated learning. By adjusting data distribution with sample weights, the authors enhance federated learning performance, achieving faster convergence and higher accuracy.

The paper [16] explores the challenges posed by data heterogeneity in federated learning environments. The authors begin by highlighting how standard federated learning algorithms perform poorly when applied to networks where data is not independently and identically distributed, leading to non-optimal results.

To address this problem, the paper introduces a novel algorithm called FedProx, which extends the traditional federated averaging (FedAvg) method. FedProx adds a proximal term to the local objective function of each participating device. This proximal term penalizes deviations from the global model, effectively regularizing the updates and mitigating the effects of data heterogeneity. The loss function establishes a balance between optimizing the local model and minimizing drift from the global model, allowing a local model to deviate from the global model only if its performance improvement justifies it. This approach prevents significant drifts that yield only minor performance gains for the local model.

The main concept of FedProx involves solving the following optimization problem at each device:

$$F_k(w) + \frac{\mu}{2}\|w - w^t\|^2 \tag{3.2}$$

Here, $F_k(w)$ represents the local function for the k-th device, w denotes the model parameters, w^t is the global model parameters at the t-th iteration, and μ is a regularization parameter that controls the influence of the proximal term.

Chapter 4

Experimental Results and Analysis

In this chapter, we present and analyze the experimental results of our study across various scenarios, some of which were discussed in previous chapters. We begin by outlining the methods and procedures used to collect data in these different contexts. Then, we detail the findings for each scenario. In the analysis section, we interpret these results and compare them to the existing research discussed earlier. This approach helps us understand the implications of our findings within the broader context of the field.

4.1 Datasets

In this research, we will utilize the CIFAR-10 dataset [14], a well-known and widely-used dataset in the field of computer vision provided by Krizhevsky et al. of the Hinton team. The CIFAR-10 dataset consists of 60,000 color images divided into 10 different classes, with each class containing 6,000 images. The images are 32×32 pixels in size, making the dataset suitable for a variety of image recognition tasks, hence our interest. The CIFAR-10 dataset includes the following 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck as shown in Figure 4.1. Each class is mutually exclusive and represents a distinct category of objects, allowing for a comprehensive evaluation of model performance across different types of images.

In our research, we will use the CIFAR-10 dataset in two ways. First, we will use all 10 classes to test the overall performance of our models across a diverse set of categories. This approach will allow us to measure how well our models generalize to a broad range of objects and scenes. Second, we will focus on only two classes: cats and dogs. This subset will allow us to investigate how our models perform with a smaller dataset and fewer categories. Also, since cats and dogs are visually similar in some aspects, this scenario will help us assess the models' classification capabilities.

The full CIFAR-10 dataset is composed of a training set of 50,000 images, evenly distributed across the 10 classes, and a test set of 10,000 images, also evenly distributed across the 10 classes. This full dataset will be used for training the models, and performing final evaluations on a diverse range of categories. For the subset of cats and dogs, the training set will consist of 10,000 images (5,000 cats and 5,000 dogs), and the test set will consist of 2,000 images (1,000 cats and 1,000 dogs). Data preprocessing is performed using the transform function in the torchvision library. The data is shuffled to prevent classes clustering when needed and normalized to enhance data integrity and reduce redundancy, with a mean and standard deviation of 0.5.

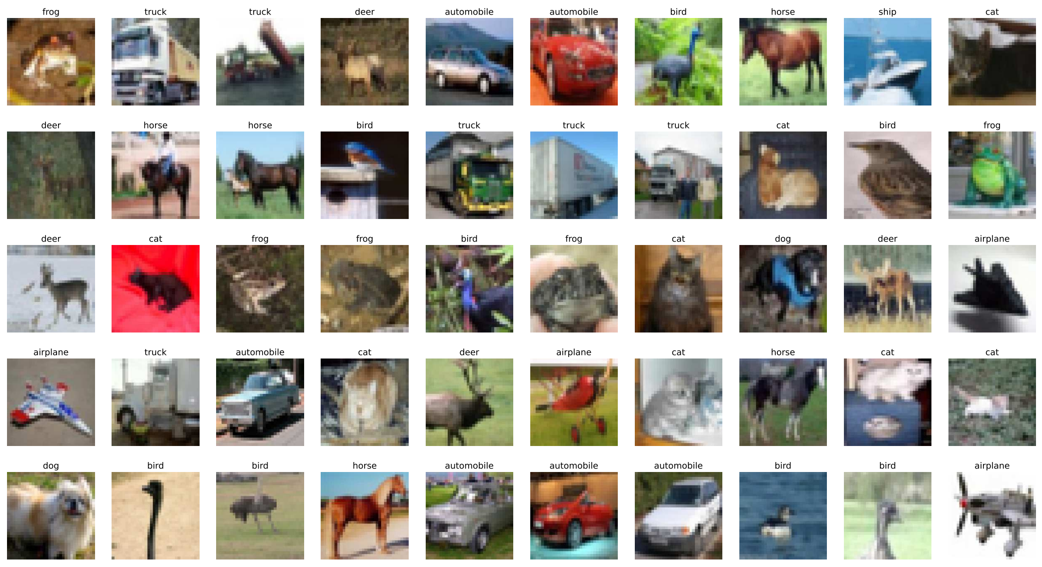


Figure 4.1: Sample image of CIFAR-10 data set.

4.2 Experimental setup

All experiments were conducted using an NVIDIA T4 GPU, known for its high performance and efficiency in machine learning tasks. The resources available included 12.7 GB of system RAM, 15.0 GB of GPU RAM, and 201.2 GB of disk storage.

The NVIDIA T4 GPU comes with 16 GB of GDDR6 memory, 320 Turing Tensor Cores, and 2,560 CUDA Cores, making it great for training and running deep learning models. It supports mixed-precision training, which helps speed up these processes. However, for our experiments, which involve a lot of tests with many peers and numerous epochs, the T4 isn't quite powerful enough. The memory and processing power are adequate for regular tasks but fall short for the large-scale and complex nature of our experiments, which involve numerous peers running for many epochs.

4.3 Model architecture

In our experiments, we initially considered ResNet18 and ShuffleNetv2 as candidate models because of their balanced performance, as it can be noticed in the model comparison table on the TorchVision website[2].

ResNet18, part of the Residual Networks (ResNet) family, is known for its simplicity and effectiveness. It works as a robust baseline model with relatively low computational and memory requirements, which makes it suitable for environments with limited resources. On the other hand, ShuffleNetv2 is designed for high efficiency on mobile devices and other low-power environments, achieving competitive accuracy with significantly reduced computation and memory usage compared to more complex models.

Understanding the detailed structure of ResNet18 is crucial because it serves as a strong foundation for many computer vision tasks. ResNet18 is widely recognized for its effectiveness and versatility in these tasks, making it important to explain how it is built internally. By exploring its architecture, we gain insights into why ResNet18 is so powerful and adaptable for different types of visual recognition problems.

The architecture of ResNet18 is composed of 18 layers, including convolutional layers and fully connected layers, organized into residual blocks. The use of residual blocks is a defining feature of ResNet18. This design helps to mitigate the vanishing gradient problem, enabling the training of deeper networks.

ResNet18 starts with a 7×7 convolutional layer, followed by a batch normalization layer and a ReLU activation function, which normalize the data and introduce non-linearity. Next, a max-pooling layer reduces the spatial dimensions. This initial sequence is followed by a series of residual blocks, each comprising two 3×3 convolutional layers. Each convolutional layer in these blocks is paired with a batch normalization layer and a ReLU activation function to ensure stable training and enhanced performance. Some blocks also include a downsampling layer to adjust feature map sizes. Prior to the final fully connected layer, a global average pooling layer is applied to further reduce spatial dimensions and minimize overfitting. The final fully connected layer outputs the classification scores for each class.

4.4 Experimental Results

This section focuses on results in different scenarios to offer an initial insight into the experimental data collected from the various cases studied. These scenarios were chosen based on the research reviewed in previous chapters, ensuring a good analysis.

In these preliminary experiments, we assessed the performance of ResNet18 [10] and ShuffleNetv2 [17] models using three distinct training methods: standalone, federated, and gossip learning. Our experiments utilized the complete CIFAR-10 dataset, which includes all 10 classes, as well as a subset containing only 2 classes (cats and dogs). Additionally, we tested both models trained from scratch and models that were fine-tuned on the ImageNet dataset[5]. For reproducibility purposes, all experiments were conducted with a random seed of 42.

By examining the performance of these models with both the complete dataset and the subset, we aim to identify which training method is most effective for different types of data. This analysis helps us understand the potential of each approach in scenarios with varying data availability and computational constraints.

Upon reviewing previous researches, we understood that achieving over 90% accuracy on datasets like CIFAR10 requires extensive training, often requiring more than 100 epochs [26]. For instance, research has shown that Shufflenetv2 models reach their peak accuracy after around 300 epochs, achieving results between 90% to 95% accuracy [26]. Additionally, studies focusing on federated learning with CIFAR have found that baseline centralized settings achieve slightly above 80% accuracy [28]. These findings highlight the importance of extended training periods for optimal performance, especially with complex datasets like CIFAR10. Given our restricted computational resources, we were unable to extend training beyond 50 epochs for the following experimentations. The reason behind choosing this number of epochs will be explained in the next section.

4.4.1 Training Hyperparameters

This section outlines the key training hyperparameters used in our experiments to evaluate the performance of ResNet18 [10] and ShuffleNetv2 [17] models across different scenarios. These hyperparameters include batch size, epochs, and learning rate, which are critical factors influencing the training process and other model performance.

In our experimental setup, we varied these hyperparameters to assess their impact on model accuracy and convergence. The choice of hyperparameters was guided by established practices and experiments done below to be sure what is best for our environment.

Batch Size

The batch size determines the number of samples processed in each iteration of training. A larger batch size can lead to faster convergence but may require more memory. We explored batch sizes ranging from 32 to 128 to find an optimal balance between efficiency and performance, see Figure 4.2.

The results show that the batch size 32 consistently demonstrates strong performance across various model groups. Its competitive accuracy, efficiency, and stability highlight its robustness in this analysis. Eventhough 32 may not always achieve the highest accuracy compared to batch sizes like 64 and 128, its close performance suggests competitiveness.

Batch size 32 represents a model with lower complexity and fewer parameters, yet it achieves commendable accuracy, indicating efficient resource utilization. This makes it a potentially more efficient choice in scenarios where there are constraints on processing power, memory, or energy consumption. Hence why we chose to continue our tests with $B = 32$.

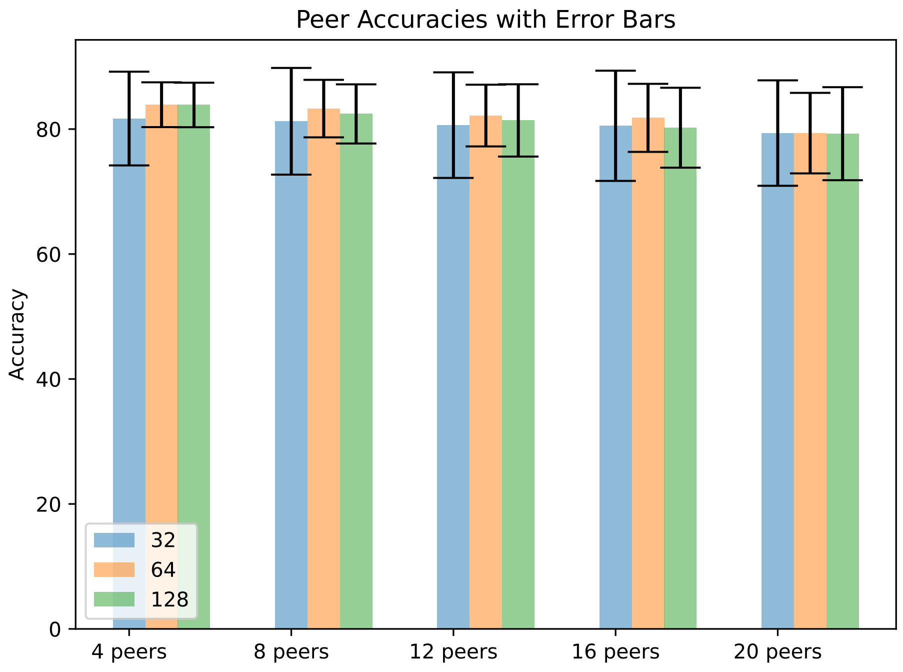


Figure 4.2: Comparing batch sizes in federated learning with ResNet18 across different numbers of peers over 50 epochs.

Epochs

Epochs indicate how many times the model goes through all the data during training. Due to our limited resources, we ran experiments using 50 epochs. We chose this to find a good balance between getting good results and considering how much time and computer resource we had available.

The Learning Rate η

The learning rate controls the step size at each iteration when updating model parameters, significantly impacting the speed and stability of convergence during training. We tested learning rates ranging from 0.1 to 0.001, as illustrated in Figure 4.3. By adjusting these parameters, we aimed to observe their impact on the model's performance. This analysis helps identify the most effective learning rate for our next experiments.

It is evident that $\eta = 0.01$ provides us with the best accuracy within 50 epochs, consistently outperforming the other options for the majority of the test duration. Therefore, we will use $\eta = 0.01$ for our subsequent tests.

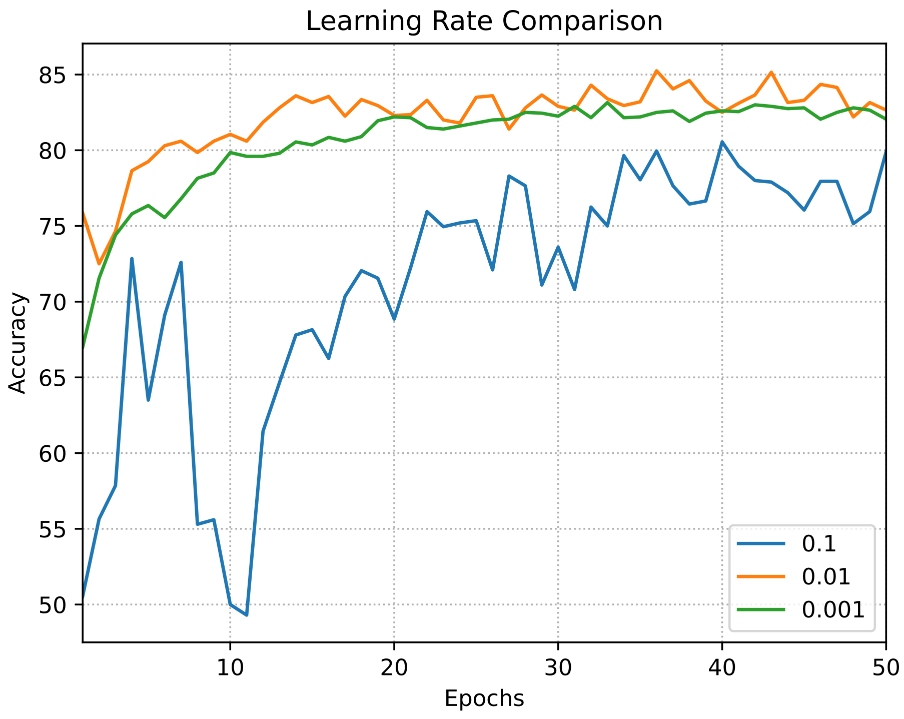


Figure 4.3: Centralized test using different learning rates for 50 epochs.

4.4.2 Centralized test

To accurately assess the outcomes of our experiments, we implemented a centralized setting where a single node is trained with the entire dataset. This setup serves as a benchmark, demonstrating the maximum achievable performance when one node has full access to all the data. These results will allow us to compare how well multiple peers can compete with this. As shown in Table 4.1, ShuffleNetv2 significantly underperforms compared to ResNet18. While we will conduct further tests with both models, it becomes evident that ResNet18 consistently delivers better results in most cases. Therefore, we mostly use ResNet18 in the subsequent experiments.

Training Type	Model	# Classes	Accuracy (%)	F1 Score (%)
Fine-tuning	ResNet18	10	84.40	84.42
Fine-tuning	ResNet18	2	81.90	82.05
Fine-tuning	ShuffleNetv2	10	73.93	73.44
Fine-tuning	ShuffleNetv2	2	77.80	77.25
From scratch	ResNet18	10	82.38	81.63
From scratch	ResNet18	2	78.80	77.84
From scratch	ShuffleNetv2	10	70.56	70.12
From scratch	ShuffleNetv2	2	73.90	72.54

Table 4.1: Performance comparison of one peer using ResNet18 and ShuffleNetv2 models, fine-tuned versus trained from scratch for 50 epochs.

4.4.3 Performance Comparison of Two Different 2 Classes Settings

In our tests, we specifically chose the two classes, cats and dogs, because they represent similar categories that can be difficult to distinguish accurately. To further illustrate this point, we conducted a test that clearly demonstrates our reasoning (Table 4.2). In the fine-tuned setting, we observed a 97.85% accuracy for cats vs ships (two distinctly different classes) compared to only 81.9% accuracy for cats vs dogs. Similarly, results from training from scratch show that cats vs ships achieved 96.5% accuracy, while cats vs dogs only reached 78.8% accuracy. The difference is even more visible in the figure 4.4. This justifies our selection of these two specific classes from the 10 available in CIFAR-10 for our upcoming experiments and validates its legitimacy.

Training Type	2 classes	Accuracy (%)	F1 Score (%)
Fine-tuning	cats vs ships	97.85	98.60
From scratch	cats vs ships	96.50	97.55
Fine-tuning	cats vs dogs	81.90	82.28
From scratch	cats vs dogs	78.80	77.72

Table 4.2: Performance comparison of one peer using two different class comparisons with ResNet18 fine-tuned versus trained from scratch for 50 epochs.

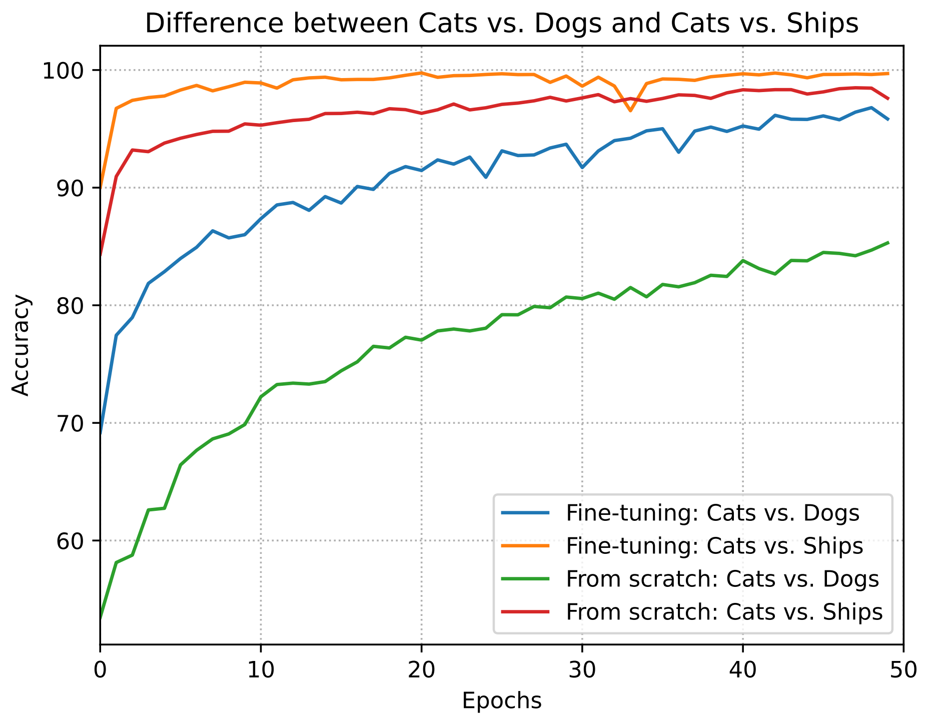


Figure 4.4: Comparison of the accuracy of the two different 2-classes choices in a fine-tuned and from scratch environment within 50 epochs.

4.5 Decentralized Learning Experiments

We now explore the performance of decentralized learning methodologies in details. Building on the preliminary results, we examine the effectiveness of federated and gossip learning approaches in a decentralized setting. These experiments aim to assess the performances of ResNet18 and ShuffleNetv2 models when trained across multiple peers, simulating real-world distributed environments.

We use both the full CIFAR-10 dataset and the subset of two classes (cats and dogs) to evaluate how data diversity and volume impact our results. By comparing the decentralized methods with traditional centralized training, we want to make in evidence the strengths and limitations of decentralized approaches, particularly under constrained computational resources and limited training epochs. This analysis offers valuable insights into the practical feasibility and potential benefits of decentralized learning, particularly in scenarios where data privacy and distribution are critical.

4.5.1 Performance Comparison: Standalone vs. Federated vs. Gossip Training

It is predictable that individual peers (standalone case) generally exhibit lower accuracy and struggle with generalization. Since each peer only has access to a portion of the dataset, they miss out on the broader data diversity needed for comprehensive training. This trend is evident in our results. For instance, in the models trained from scratch (Table 4.3), standalone ResNet18 models achieved accuracies of 69.94% with 4 peers and 56.49% with 10 peers when using all 10 classes. Similarly, standalone ShuffleNetv2 models achieved accuracies of 58.13% with 4 peers and 48.15% with 10 peers under the same conditions. These values show the limitations of standalone training, where models lack the benefit of shared knowledge from other peers, resulting in lower overall performance.

The federated learning approach demonstrated significantly better performance compared to standalone training. For instance, federated ResNet18 models achieved an accuracy of 85.75% with 4 peers and 84.36% with 10 peers when fine-tuned (Table 4.4). The ShuffleNetv2 models showed a similar trend, with federated learning achieving 71.47% accuracy with 4 peers and 66.20% with 10 peers. This improvement underscores the advantage of federated learning, where models benefit from the collaborative training process, leading to enhanced generalization and accuracy. In this case, we used the system of communication rounds and used every epoch as one to average every model's weights with each other.

As expected, gossip learning provided us with efficient outcomes, comparable to those of federated learning. However, the extent of its effectiveness surpassed our expectations. In all cases, gossip learning outperformed standalone training, showing superior performance. It even outperformed federated learning in many cases, especially the 2 classes case with ResNet18. Notably, in the case of fine-tuned models, the gossip ResNet18 achieved an accuracy of 83.29% with 4 peers, surpassing the 82.29% achieved by federated learning (Table 4.4). Similarly, with 10 peers, it achieved 82.53%, outperforming the 82.09% attained by federated learning. This improvement suggests that gossip learning has a small advantage with smaller datasets, which aligns with real-world scenarios.

Initially, we aggregated peers every epoch, similar to federated learning, but this approach provided us with poor results, even worse than standalone training. This was expected since gossip learning is designed to aggregate one peer with another, rather than all peers simultaneously as federated learning does. Consequently, we revised the algorithm to a more efficient version that properly aggregates gossip models. In this revised method, gossip models were aggregated more frequently, being combined different times within each epoch. Aggregation is based on the age, t, which starts at 0 for all peers and increases by the maximum age of the peer and its randomly selected partner. Each peer is assigned a random partner in each iteration within the epoch, and the peer's age is updated to the maximum of both ages incremented by one. We chose a fully connected network for these tests where every peer is connected with another giving the possibility to aggregate with any peer in the system.

Training Type	**Model**	**# Peers**	**# Classes**	**Accuracy (%)**	**F1 Score (%)**
Standalone	ResNet18	4	10	69.94	69.27
Federated	ResNet18	4	10	79.90	79.55
Gossip	ResNet18	4	10	79.16	75.32
Standalone	ResNet18	10	10	56.49	55.10
Federated	ResNet18	10	10	69.55	68.05
Gossip	ResNet18	10	10	67.21	64.43
Standalone	ResNet18	4	2	69.14	66.62
Federated	ResNet18	4	2	76.6	75.73
Gossip	ResNet18	4	2	76.88	77.44
Standalone	ResNet18	10	2	62.76	63.12
Federated	ResNet18	10	2	73.5	75.94
Gossip	ResNet18	10	2	71.44	61.05
Standalone	ShuffleNetv2	4	10	58.13	56.24
Federated	ShuffleNetv2	4	10	64.22	63.73
Gossip	ShuffleNetv2	4	10	79.59	76.12
Standalone	ShuffleNetv2	10	10	48.15	45.73
Federated	ShuffleNetv2	10	10	56.88	55.11
Gossip	ShuffleNetv2	10	10	70.78	68.06
Standalone	ShuffleNetv2	4	2	62.24	63.87
Federated	ShuffleNetv2	4	2	63.51	68.55
Gossip	ShuffleNetv2	4	2	74.45	74.37
Standalone	ShuffleNetv2	10	2	57.77	46.03
Federated	ShuffleNetv2	10	2	62.37	58.20
Gossip	ShuffleNetv2	10	2	69.76	66.75

Table 4.3: Comparison of ResNet18 and ShuffleNetv2 performance across different training methods from scratch for 50 epochs.

Training Type	**Model**	**# Peers**	**# Classes**	**Accuracy (%)**	**F1 Score (%)**
Standalone	ResNet18	4	10	77.51	76.25
Federated	ResNet18	4	10	85.75	85.63
Gossip	ResNet18	4	10	82.45	77.72
Standalone	ResNet18	10	10	71.22	69.73
Federated	ResNet18	10	10	84.36	84.81
Gossip	ResNet18	10	10	76.85	72.16
Standalone	ResNet18	4	2	78.13	78.22
Federated	ResNet18	4	2	82.15	82.15
Gossip	ResNet18	4	2	83.29	84.25
Standalone	ResNet18	10	2	75.2	74.04
Federated	ResNet18	10	2	82.09	82.16
Gossip	ResNet18	10	2	82.53	84.29
Standalone	ShuffleNetv2	4	10	67.44	65.38
Federated	ShuffleNetv2	4	10	71.47	70.73
Gossip	ShuffleNetv2	4	10	83.14	78.51
Standalone	ShuffleNetv2	10	10	59.23	58.11
Federated	ShuffleNetv2	10	10	66.20	65.06
Gossip	ShuffleNetv2	10	10	76.81	70.26
Standalone	ShuffleNetv2	4	2	73.88	74.72
Federated	ShuffleNetv2	4	2	75.01	74.63
Gossip	ShuffleNetv2	4	2	82.71	82.55
Standalone	ShuffleNetv2	10	2	68.92	71.98
Federated	ShuffleNetv2	10	2	70.02	69.47
Gossip	ShuffleNetv2	10	2	81.31	78.18

Table 4.4: Comparison of fine-tuned ResNet18 and ShuffleNetv2 performance across different training methods for 50 epochs.

The main takeaway from these preliminary experiments is that gossip learning, while capable of enhancing model performance over federated training, does not consistently do so. Gossip learning aims to match the performance of federated learning in some cases (2 classes) while not overtaking it and surpasses it in some other instances. Despite this, gossip learning presents a truly decentralized approach, addressing our primary concern of privacy, which gives it more importance. Given the balanced dataset, reporting balanced accuracy is unnecessary, but these findings are important for understanding the dynamics of different training methodologies in distributed learning environments.

Overall, both gossip and federated learning proved to be the most effective methods in our experiments, significantly enhancing accuracy and generalization compared to the standalone approach. We also compared the performance of fine-tuned models to those trained from scratch, clearly demonstrating that the fine-tuned models significantly outperform the ones trained from scratch. As previously noted, ResNet18 consistently outperforms ShuffleNetv2 in all scenarios, whether fine-tuned or trained from scratch. Given these results, it is clear that further testing with ShuffleNetv2 is unnecessary, and we will focus exclusively on ResNet18 in our future experiments.

4.5.2 Unbalanced Partition Sizes

The challenge of unbalanced data distribution among peers is a common occurrence in a real-world setting. It represents the situation where not all peers possess similar amounts of data.

We created this unbalanced data scenario by partitioning a dataset into subsets using an exponentially distributed proportions approach. Initially, we generated random values uniformly distributed between 0 and 1 and transformed them into an exponential distribution by taking the negative logarithm. After normalizing these values to ensure their sum equals 1, we multiplied the normalized proportions by the total dataset length to determine the size of each subset. Finally, we iterated through these sizes to create subsets of indices corresponding to each size.

We conducted tests on standalone, federated, and gossip approaches, exploring cases involving 4 and 10 peers, using both all classes of CIFAR-10 and the binary classification of cats vs dogs. Our findings consistently demonstrate federated learning as the top performer in this context, followed by gossip learning, with standalone performance ranking the lowest (Table 4.5). This outcome is as expected, since standalone peers operating independently make it that the presence of peers with limited data drags down the overall accuracy of the group.

Training Type	**Model**	**# Peers**	**# Classes**	**Accuracy (%)**	**F1 Score (%)**
Standalone	ResNet18	4	10	56.49	69.21
Federated	ResNet18	4	10	83.49	80.53
Gossip	ResNet18	4	10	75.31	68.79
Standalone	ResNet18	10	10	60.97	69.93
Federated	ResNet18	10	10	85.23	81.61
Gossip	ResNet18	10	10	77.24	75.21
Standalone	ResNet18	4	2	70.5	42.84
Federated	ResNet18	4	2	75.79	72.66
Gossip	ResNet18	4	2	74.98	73.17
Standalone	ResNet18	10	2	70.09	74.52
Federated	ResNet18	10	2	79.61	80.07
Gossip	ResNet18	10	2	78.27	69.41

Table 4.5: Comparison of fine-tuned ResNet18 performance across different training methods for 50 epochs in an unbalanced dataset setting.

An important challenge we encountered was the varying training durations for each peer within an epoch, conversely with previous cases where all peers trained for the same duration. This difference has minimal impact on federated and standalone cases. In federated learning, models aggregate after each epoch, by which time all models have completed their training. Conversely, standalone peers do not aggregate, eliminating any issues related to training duration discrepancies.

However, in the gossip environment, this scenario poses a significant challenge. The age t of each peer starts to diverge over time, with some peers having limited data and thus fewer opportunities for aggregation, while others have more chances to aggregate, enhancing their efficiency and getting them a higher age value. This difference in age gives an advantage to certain peers during aggregation, favoring their weights when aggregating, which complicates the dynamics of the gossip approach.

4.5.3 Non-IID Data

Non-Independent and Identically Distributed (Non-IID) data presents a significant challenge in decentralized learning environments. This scenario arises when the distribution of data across different peers is not uniform, meaning that each peer's dataset varies in class distribution and feature space, which makes the problem of classification harder due to the lack of diversity while training the models.

In our study, we explored the impact of Non-IID data on the performance of federated, gossip, and standalone learning approaches using the CIFAR-10 dataset. To create a highly skewed distribution, we partitioned the dataset so that each peer received data from a limited subset of classes. Specifically, we employed a sharding technique, dividing the dataset into $3 \times n$ shards, where n represents the number of peers. These shards were then randomly assigned to different peers, ensuring that most peers had data from at least two classes, enabling them to classify the data accurately.

Our findings, detailed in Table 4.6, reveal that federated learning continues to outperform both gossip and standalone methods under Non-IID conditions, with one case where the gossip technique is able to slightly outperform it. This is attributed to the federated approach's ability to aggregate diverse models, thereby mitigating the skewness in individual peers' data. Conversely, the gossip approach, while slightly more robust than standalone, struggles with the lack of diversity in the data it aggregates, especially noticeable in the 10 classes cases, leading to lower overall performance. The standalone method, as expected, performed the worst, as the lack of data sharing and aggregation limits its ability to generalize. Figure 4.5 visually demonstrates the performance disparities among these methods, emphasizing the federated model's superior ability to handle Non-IID data distributions.

Training Type	**# Classes**	**# Peers**	**Accuracy (%)**	**F1 Score (%)**
Federated	10	4	61.76	56.21
Federated	10	10	44.49	25.41
Gossip	10	4	50.35	23.63
Gossip	10	10	25.69	08.86
Standalone	10	4	38.19	22.52
Standalone	10	10	26.76	07.74
Federated	2	4	83.29	80.05
Federated	2	10	83.58	83.21
Gossip	2	4	83.71	83.24
Gossip	2	10	77.22	79.41
Standalone	2	4	64.49	60.18
Standalone	2	10	67.13	73.66

Table 4.6: Comparison of fine-tuned ResNet18 across different training types within 50 epochs using all classes.

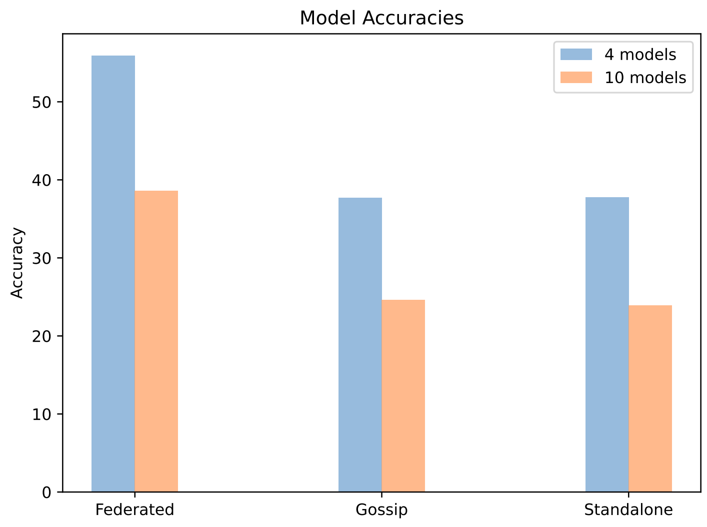


Figure 4.5: Comparison of fine-tuned ResNet18 performance accross different training methods using all classes in a non-IID environment within 50 epochs.

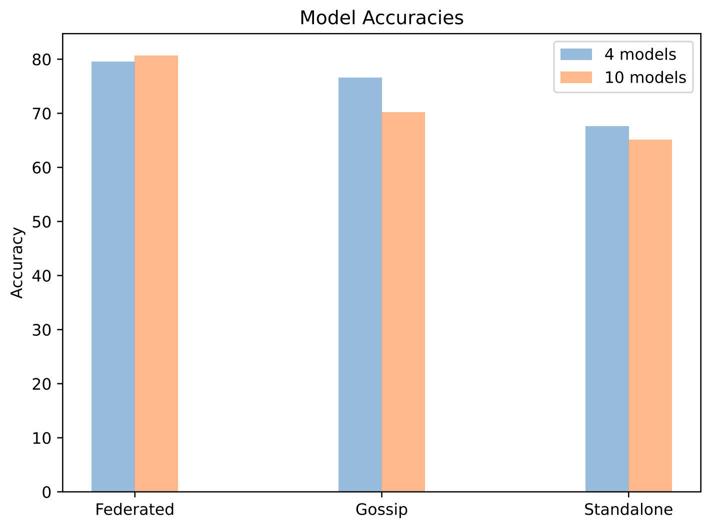


Figure 4.6: Comparison of fine-tuned ResNet18 performance accross different training methods using 2 classes in a non-IID environment within 50 epochs.

4.5.4 Extensive Experiments on the Unbalanced Data Scenario

In our extensive experiments, we incrementally added two peers at a time, up to a total of 20 peers, and tested federated, gossip, and standalone methods. Initially, we applied this process to an unbalanced dataset with only two classes, chosen for its simplicity and quick evaluation potential. Despite the limited scope, the results were compelling: federated learning consistently outperformed the other methods, followed by gossip learning, with standalone methods lagging behind, as shown in Figure 4.7. These findings highlight the robustness of federated learning, even in a constrained experimental setup.

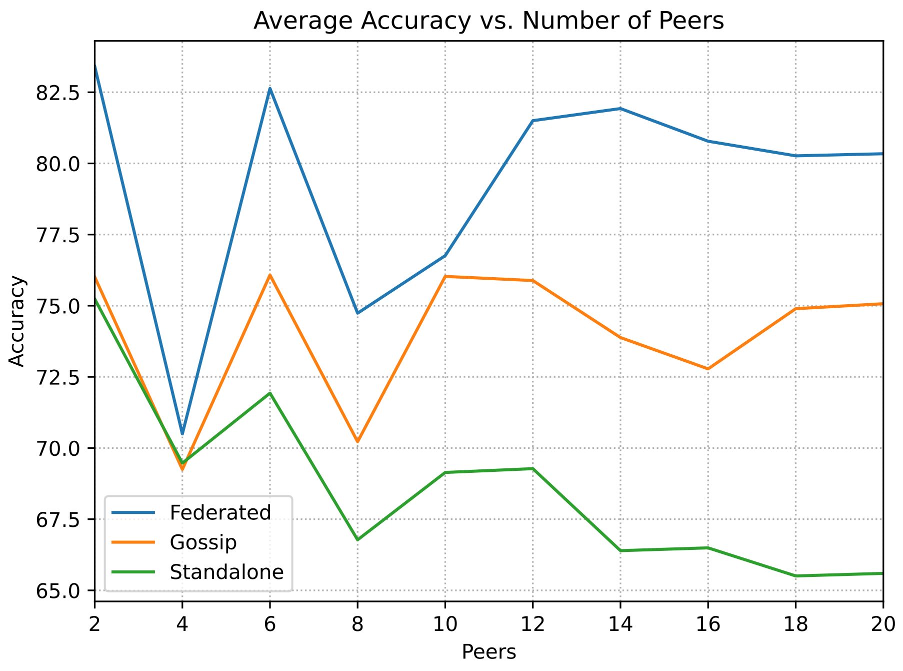


Figure 4.7: Comparison of fine-tuned ResNet18 final accuracy accross different training methods using 2 classes in an unbalanced environment in term of number of peers within 50 epochs.

We conducted additional tests to ensure precise results by increasing the number of peers by four in each step until reaching 20, repeating each test scenario three times for reliable average accuracy. Instead of using a line graph, we opted for a detailed bar plot presentation that includes error bars to depict the variability of each technique, as shown in Figure 4.8. Interestingly, the gossip method showed increasing stability with additional peers. In contrast, while the federated method also demonstrated improved stability with more peers, this trend was not as pronounced as with the gossip method. Standalone performance remained consistent with minimal variance across all tested scenarios.

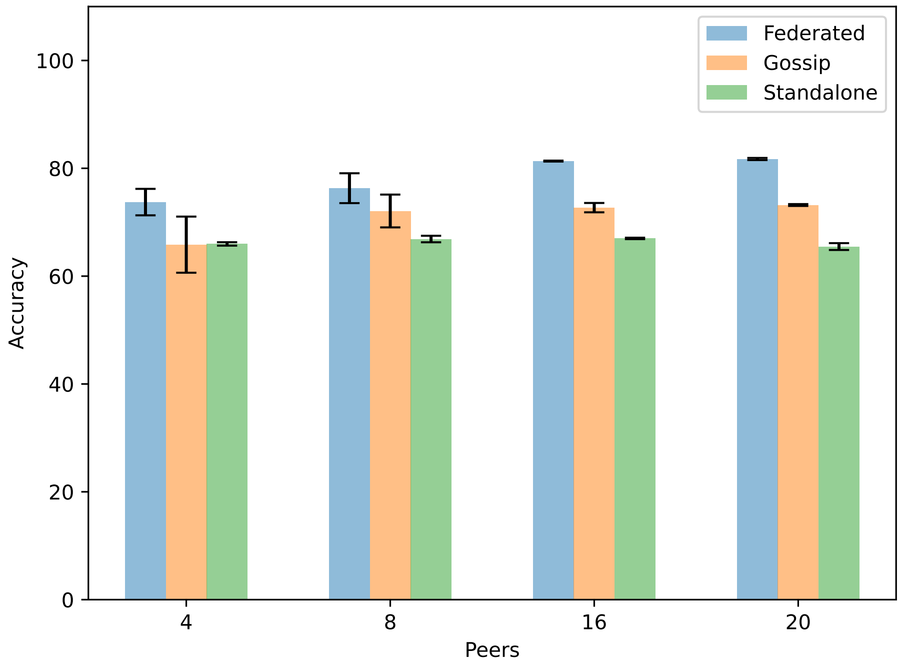


Figure 4.8: Comparison of fine-tuned ResNet18 final accuracy accross different training methods using 2 classes in an unbalanced environment in term of number of peers within 50 epochs.

To gain a clearer understanding of these values, we also included a line graph that represents accuracy over epochs. The graph features the 75th and 25th percentiles, shown with faded blue and darker faded blue, respectively, allowing us to observe the evolution of each learning method over time. We were pleasantly surprised by the strong performance of the federated approach, as its graph resembled a typical line graph, with the 25th and 75th percentiles closely aligning with the median (Figure 4.9). In contrast, the standalone method produced consistent results but showed no significant improvement over time. However, the gossip learning method demonstrated better performance, with the range of the 75th percentile decreasing over time and the 25th percentile already close to the median, unlike the standalone method, as seen in the Figure 4.10.

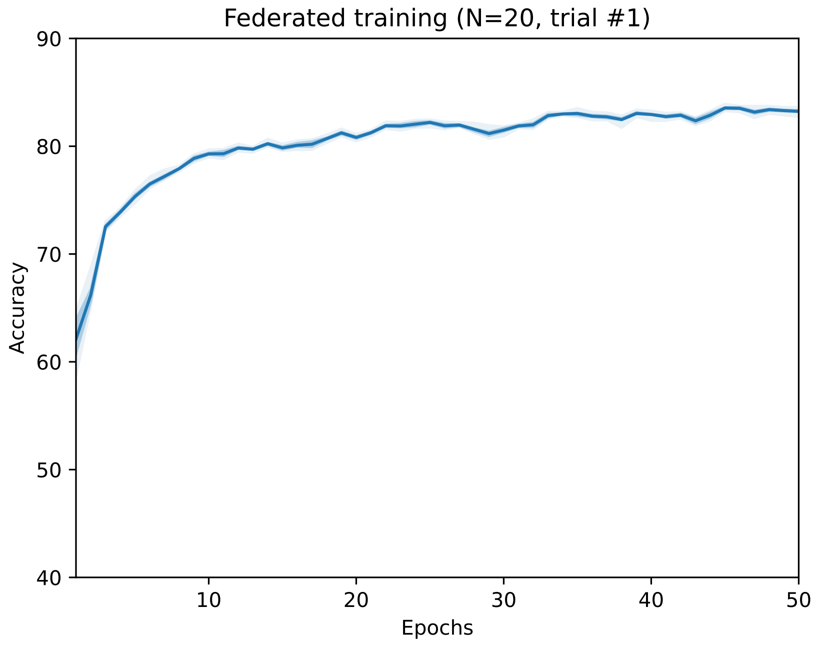
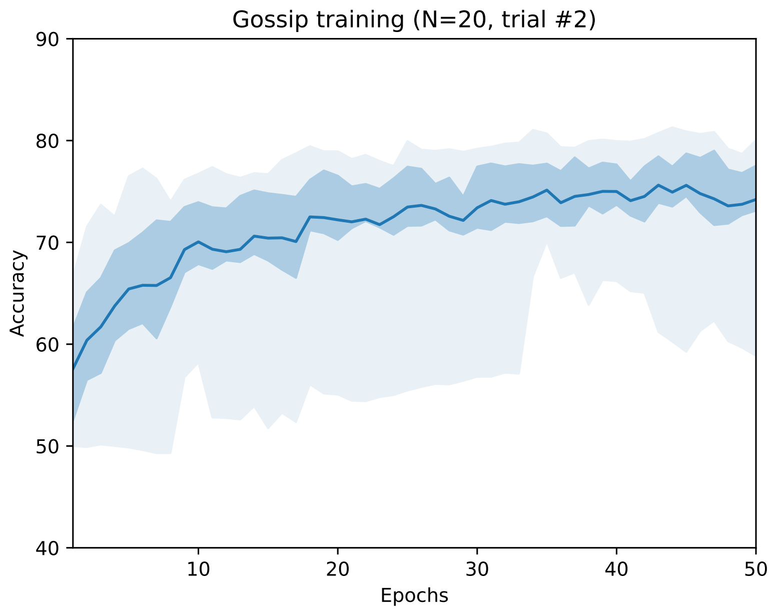


Figure 4.9: Line graph of fine-tuned ResNet18 accuracy using the federated training method with 2 classes in an unbalanced environment using 20 peers within 50 epochs.



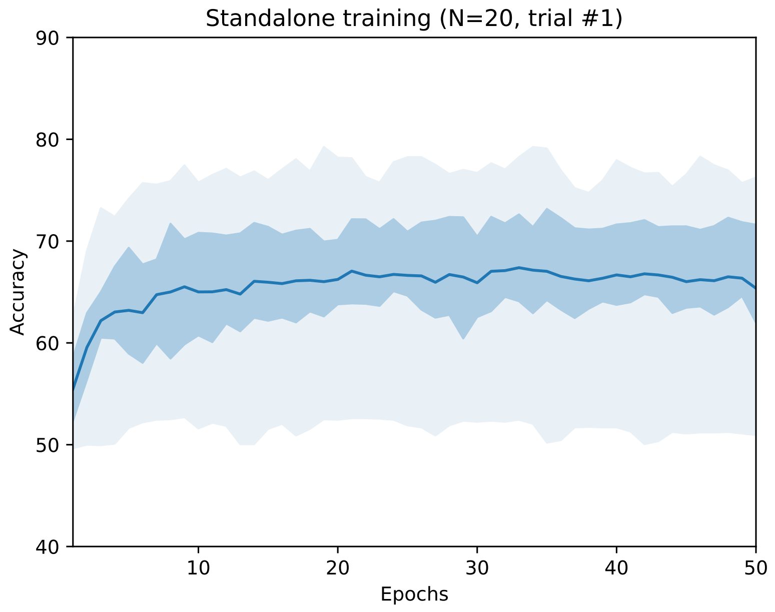


Figure 4.10: Line graph of fine-tuned ResNet18 accuracy using gossip and standalone training methods with 2 classes in an unbalanced environment using 20 peers within 50 epochs.

4.5.5 Impact of Adjacency Matrix

In the next phase of our tests, we focused on modifying the adjacency matrix for gossip learning. We utilized different types of graph structures, including Barabási-Albert and Erdős-Rényi models, to explore their impact on performance. The Barabási-Albert model generates a scale-free network where some nodes become highly connected, simulating real-world networks. In contrast, the Erdős-Rényi model creates a random graph where each edge is included with a fixed probability, resulting in a more uniformly distributed structure.

Initially, we performed two sets of tests using average degrees of 10 and 8, respectively, over 30 epochs, seen in the Table 4.7. These variations allowed us to assess how connectivity affects the efficiency and accuracy of gossip learning, leading us to conclude that an average degree of 10 yields the best results.

We conducted experiments with 30 peers using the ResNet18 architecture on the CIFAR-10 dataset for 50 epochs. The accuracy results of fully connected gossip, Barabási-Albert gossip, and Erdős-Rényi gossip were compared against those of federated and standalone learning methods seen in the Table 4.8.

Training Type	**# Peers**	**Accuracy (%)**	**F1 Score (%)**
Federated	30	43.75	41.22
Standalone	30	26.61	54.21
Normal gossip	30	65.52	66.05
Gossip barabasi_albert (1)	30	61.74	64.09
Gossip barabasi_albert (2)	30	61.51	60.24
Gossip erdos_renyi (1)	30	57.49	65.04
Gossip erdos_renyi (2)	30	48.69	41.71

Table 4.7: Comparison of fine-tuned ResNet18 across different training types within 30 epochs using all classes.

(1): Barabasi Albert: Average degree of 10
(2): Barabasi Albert: Average degree of 8

Training Type	# Peers	Accuracy (%)	F1 Score (%)
Federated	30	51.31	40.21
Standalone	30	24.09	56.30
Fully Connected gossip	30	65.10	67.08
Gossip Barabási-Albert	30	63.20	56.41
Gossip Erdős-Rényi	30	62.44	46.32

Table 4.8: Comparison of fine-tuned ResNet18 across different training types within 50 epochs using all classes and an average of three test each.

The results presented in Table 4.8 reflect the average accuracy across three runs for each training method (federated, standalone, and the three gossip methods), conducted over 50 epochs. This averaging ensures the stability and reliability of the reported information. In this test, gossip learning methods outperformed both federated learning by over 13% and standalone learning by 40% in terms of accuracy. Additionally, the Barabási-Albert method demonstrated slightly better performance compared to the Erdős-Rényi method, where both had an average degree of 10.

We initially expected federated learning to outperform all other methods. However, the results were surprising, with gossip methods showing a significant advantage. The presence of multiple partitions increased the likelihood of including low-accuracy peers (as seen in the standalone training results), which negatively impacted the final results for federated learning. In contrast, gossip learning methods consider the age of the peers, making it more likely to retain weights from better-performing peers, thus enhancing overall accuracy.

4.5.6 Euclidean Distance

In our experiment, we utilize the Euclidean distance as a standard metric to measure the similarity or distance between data points. We assess the distance between feature vectors extracted from the CIFAR-10 dataset using a fine-tuned ResNet18 model. This test is important because it gives us a clear and measurable way to see how well the model is doing, especially when it comes to accuracy and how well it groups similar things together.

We calculate the Euclidean distance over the course of 50 epochs using 30 peers, repeating this process three times and averaging the results for greater reliability. This approach allows us to analyze the model's convergence behavior and refine its predictive capabilities. The outcomes of this test are crucial for evaluating the model's classification accuracy, as they indicate how closely the predicted feature vectors align with the actual data points.

By using Euclidean distance here, we can clearly see how the model learns over time. This metric allows us to compare different network topologies and evaluate the performance of our method in each scenario. In our experiment, we evaluated three network topologies: the Barabasi-Albert network, the Erdos-Renyi model, and the fully connected gossip learning approach. The results demonstrated that fully connected gossip learning consistently achieved the smallest Euclidean distances, which was expected due to the ease of communication among all connected models. The Barabasi-Albert network also performed well, slightly lagging behind the fully connected results, followed by the Erdos-Renyi network. These findings indicate that our gossip learning method is effective with both Barabasi-Albert and Erdos-Renyi network topologies, even with a relatively large number of peers, such as 30 (Figure 4.11).

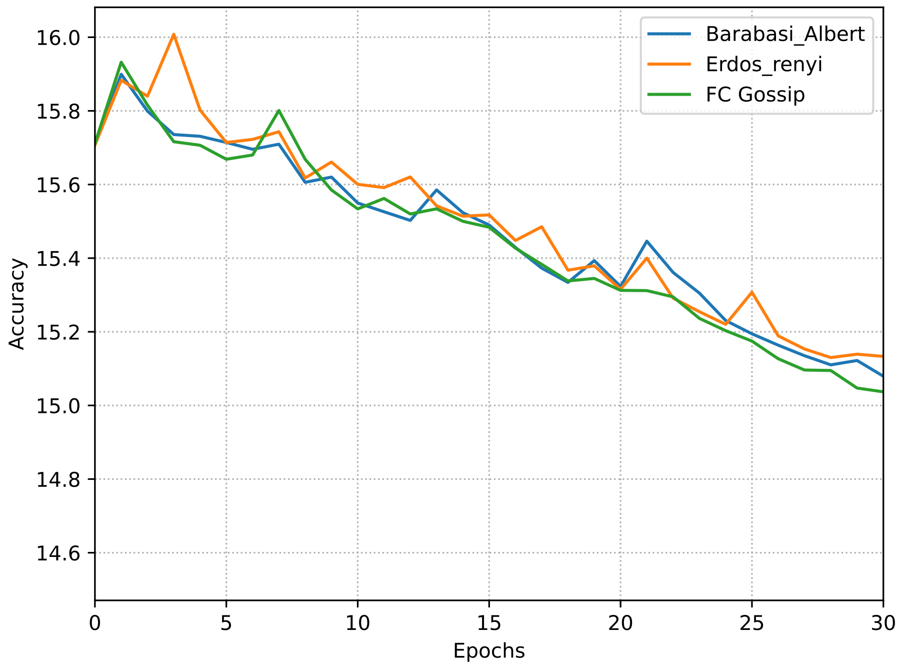


Figure 4.11: Line graph comparison of euclidean distance of different network models using 30 peers within 50 epochs.

These findings confirmed that training is feasible across various network topologies using multiple models. To better differentiate between the network topologies, more models should be used, since the slight differences in results might come from the limited number of model exchanges. Therefore, adding more peers and training them over more epochs would provide us with more conclusive results. However, implementing these experiments is challenging due to test limitations. Despite this, our approach still provides valuable insights into the practical applications of our method across different network topologies.

Chapter 5

Conclusion

We studied how effective decentralized fine-tuning is using federated and gossip learning across different scenarios, comparing them with standalone methods. Our experiments consistently showed that decentralized fine-tuning performed better than standalone approaches in all situations, including unbalanced partition sizes, non-IID data, and scenarios with numerous peers sharing data among themselves. However, neither federated nor gossip learning consistently outperformed the other. Their performance varied depending on specific factors like the number of peers or classes involved.

Additionally, our tests revealed that fine-tuning significantly outperforms learning from scratch, with accuracy improvements of up to 10% in many cases (Tables 4.3 and 4.4). This advantage was evident across all methods, including standalone, federated, and gossip learning.

Our experiments showed that fine-tuning using decentralized methods were effective even with just 50 epochs, achieving over 80% accuracy in most cases. In scenarios with unbalanced partition sizes, federated learning achieved 85.23% accuracy when using 10 peers for the entire dataset (Table 4.5). The results for non-IID data varied, achieving promising accuracy above 80% for the 2-class scenario but falling below 62% for the 10-class scenario. This drop in accuracy could be attributed to the limited training opportunities, which might improve with better resources allowing us to train for more epochs.

We also conducted an experiment involving 30 peers, where gossip learning significantly surpassed federated learning. This was because gossip learning aggregates weights based on peers' age, avoiding excessive influence from peers with lower accuracy, unlike federated learning. In this case, gossip learning achieved over 40% higher accuracy than the standalone approach and nearly 15% more than federated learning. Although the final accuracy was lower than in other experiments, this was likely due to our limited number of epochs. These results highlight the potential of decentralized fine-tuning techniques to improve machine learning model performance across various real-world applications.

5.1 Future Works

Unlike federated learning, which has been extensively studied with different aggregation methods and in diverse scenarios using CIFAR10, gossip learning has not yet been thoroughly experimented with, to our knowledge. This makes our work an intriguing area for future researchers to explore.

To increase our understanding of this research, future studies should focus on conducting experiments with a significantly higher number of epochs, ideally exceeding 300 epochs. The limited number of epochs in our study was also a major constraint that impacted our findings. Replicating these tests with a larger number of peers would better simulate the complexity of real-world applications, providing clearer insights into the effectiveness of these methods.

Addressing these limitations will allow future research to more thoroughly validate and expand upon the promising findings of this study, providing deeper insights into the practical applications of decentralized fine-tuning for machine learning models.

Bibliography

[1] Jinyu Chen et al. “FedTune: A Deep Dive into Efficient Federated Fine-Tuning with Pre-trained Transformers”. In: *arXiv preprint arXiv:2211.08025* (2022). URL: `https://arxiv.org/abs/2211.08025`.

[2] TorchVision Contributors. *TorchVision Models*. Accessed: 2024-06-27. 2023. URL: `https://pytorch.org/vision/stable/models.html`.

[3] Yutong Dai et al. *Tackling Data Heterogeneity in Federated Learning with Class Prototypes*. 2023. arXiv: `2212.02758 [cs.LG]`.

[4] Jeffrey Dean et al. “Large scale distributed deep networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Curran Associates Inc. USA, Dec. 2012, pp. 1223–1231.

[5] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255. URL: `https://ieeexplore.ieee.org/abstract/document/5206848/`.

[6] Haoyu Dong et al. *Table Pre-training: A Survey on Model Architectures, Pre-training Objectives, and Downstream Tasks*. 2022. arXiv: `2201.09745 [cs.CL]`.

[7] European Commission. *General Data Protection Regulation (GDPR)*. Accessed: April 10, 2024. 2018. URL: `https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/dataprotection/2018-reform-eu-data-protection-rules`.

[8] Mathieu Germain et al. “MADE: Masked Autoencoder for Distribution Estimation”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July

2015, pp. 881–889. URL: `https://proceedings.mlr.press/v37/germain15.html`.

[9] Lodovico Giaretta and Sarunas Girdzijauskas. "Gossip Learning: Off the Beaten Path". In: *2019 IEEE International Conference on Big Data (IEEE Big Data 2019)*. IEEE. 2019. URL: `%5E45%5E`.

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: `1512.03385 [cs.CV]`.

[11] István Hegedűs, Gábor Danner, and Márk Jelasity. "Decentralized learning works: An empirical comparison of gossip learning and federated learning". In: *Journal of Parallel and Distributed Computing* 148 (2021), pp. 109–124. URL: `https://bit.ly/3j4R7JK`.

[12] István Hegedűs et al. "Robust Decentralized Low-Rank Matrix Decomposition". In: *ACM Transactions on Intelligent Systems and Technology* 7.4 (2016), p. 62. URL: `https://www.inf.u-szeged.hu/~berta/publications/tist15.pdf`.

[13] Jakub Konečný et al. "Federated learning: Strategies for improving communication efficiency". In: *Private Multi-Party Machine Learning (NIPS 2016 Workshop)*. 2016.

[14] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[15] Qinbin Li et al. *Federated Learning on Non-IID Data Silos: An Experimental Study*. 2021. arXiv: `2102.02079 [cs.LG]`.

[16] Tian Li et al. *Federated Optimization in Heterogeneous Networks*. 2020. arXiv: `1812.06127 [cs.LG]`. URL: `https://arxiv.org/abs/1812.06127`.

[17] Ningning Ma et al. *ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design*. 2018. arXiv: `1807.11164 [cs.CV]`.

[18] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. 20–22 Apr 2017. PMLR. Fort Lauderdale, FL, USA, Apr. 2017, pp. 1273–1282.

[19] Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[20] Calvin Newport, Alex Weaver, and Chaodong Zheng. *Asynchronous Gossip in Smartphone Peer-to-Peer Networks*. 2021. arXiv: 2102.06804 [cs.DC].

[21] Hung Nguyen, Peiyuan Wu, and Morris Chang. *Federated Learning for distribution skewed data using sample weights*. 2024. arXiv: 2401.02586 [cs.LG].

[22] Róbert Ormándi, István Hegedűs, and Márk Jelasity. "Gossip learning with linear models on fully distributed data". In: *arXiv preprint arXiv:1109.1396* (2011). URL: https://arxiv.org/abs/1109.1396.

[23] Róbert Ormándi, István Hegedűs, and Márk Jelasity. "Gossip learning with linear models on fully distributed data". In: *Concurrency and Computation: Practice and Experience* 25.4 (May 2012), pp. 556–571. ISSN: 1532-0634. DOI: 10.1002/cpe.2858. URL: http://dx.doi.org/10.1002/cpe.2858.

[24] Nima Tajbakhsh et al. "Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?" In: *IEEE Transactions on Medical Imaging* 35.5 (2016), pp. 1299–1312. DOI: 10.1109/TMI.2016.2535302.

[25] Lingling Xu et al. *Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment*. 2023. arXiv: 2312.12148 [cs.CL].

[26] Longqing Ye. *AugShuffleNet: Communicate More, Compute Less*. 2022. arXiv: 2203.06589 [cs.CV].

[27] Aston Zhang et al. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG].

[28] Yue Zhao et al. "Federated Learning with Non-IID Data". In: (2018). DOI: 10.48550/ARXIV.1806.00582. URL: https://arxiv.org/abs/1806.00582.

[29] Jingyuan Zhu et al. *DomainStudio: Fine-Tuning Diffusion Models for Domain-Driven Image Generation using Limited Data*. 2024. arXiv: 2306.14153 [cs.CV].