CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Studying Imperfect Communication In Distributed Optimization Algorithm

Master's thesis in Computer science and engineering

Swati Math, Madhumitha Venkatesan

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Studying Imperfect Communication In Distributed Optimization Algorithm

Swati Math **and** Madhumitha Venkatesan

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

A Chalmers University of Technology

Supervisor: Ashkan Panahi, DS&AI
Examiner: Ashkan Panahi, DS&AI

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2024

A Chalmers University of Technology

Swati Math
Madhumita Venkatesan
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Distributed optimization methods are essential in machine learning, especially when data is distributed across multiple nodes or devices. These algorithms enable effective model training without data consolidation, improving privacy and reducing communication costs. However, their performance is greatly influenced by the quality of communication, which may degrade due to factors such as quantization and erasure. Quantization, which involves estimating values during transmission, can result in loss of information and requires strategic optimization to manage distortion and communication expenses. Similarly, erasure causes loss of transmitted information, leading to delays in convergence ,increased energy usage. This study explores how communication imperfections affect the performance of distributed optimization algorithms, emphasizing convergence rates, scalability, and overall efficiency. The research examines how quantization and erasure impact different distributed architectures like Federated Learning and push-pull gradient methods under different network topologies and suggests ways to reduce their effects.

Keywords: Distributed optimization, machine learning, quantization, erasure, convergence, communication overhead, scalability, Federated Learning, push-pull gradient methods, distributed systems.

Acknowledgements

We wish to express our gratitude to Prof. Ashkan Panahi, who supervises us at the Department of Computer Science and Engineering at Chalmers University. His steadfast support, wise advice, and priceless expertise have been crucial to the accomplishment of our project. We really appreciate his support and his kind sharing of his extensive knowledge on distributed optimization techniques, which have improved our comprehension and enabled us to successfully navigate the complexities of this difficult subject. We could not have completed this assignment without his commitment and guidance.

Swati Math, Gothenburg,
Madhumitha venkatesan, Gothenburg,
2024-11-08

Contents

List of Figures

1
Introduction

Distributed optimization algorithms are designed to solve the optimization problems where the computation is spread across multiple nodes or clients or devices. These algorithms are widely employed in large-scale systems such as machine learning, where decentralized processing is critical for handling huge amounts of data and model parameters. By distributing the workload across several agents, such algorithms ensure scalable and efficient solutions without the need for a centralized systems. The workload is distributed across the nodes or clients, allowing for parallel processing, which increases computational efficiency and saves training time. This parallel technique enables simultaneous computations on various sections of the data, resulting in faster convergence and shorter overall processing time. Beyond machine learning, distributed optimization also plays a crucial role in other large-scale systems, such as networked systems where it optimizes resource allocation, routing, and load balancing, assuring efficient network performance. Distributed optimization is particularly important in sensor networks for tasks such as data gathering, target tracking, and environmental monitoring. It allows sensors to collectively process data and make judgments [1].

In traditional centralized systems, a single central server has access to the entire dataset and is responsible for performing all computations. While this approach works well for smaller datasets, it quickly becomes inefficient for large-scale systems due to scalability issues and communication overhead. Centralized methods also create a single point of failure, meaning that if the central server crashes or becomes overloaded, the entire system can be compromised.

These limitations highlight the importance of decentralized approaches in large-scale systems. In decentralized system no single node has access to the entire data set; instead, each node operates on a local data subset. To ensure a global solution, nodes must communicate with one another; However, developing effective communication protocols is critical, as excessive communication might cause bottlenecks, reducing convergence.

To minimize the communication overhead quantization is used, which is the process of reducing the precision of the data being communicated between nodes or clients. However, while this technique can optimize bandwidth (reducing communication overhead) and speed up training, it may introduce small errors called quantization error which affects the model performance. In addition to quantization errors, erasure or message loss may occur. This means that information such as model updates

or gradient information are lost during transmission between nodes. This data loss can occur due to various reasons, such as network issues, corrupted data packets, or dropped connections. This can slow down the convergence process or lead to inaccurate updates in the model, as missing messages prevent nodes from fully synchronizing their local states.

The convergence rate in distributed optimization is not only influenced by the number of nodes or clients involved and the desired level of accuracy but also on the structure and nature of the network over which nodes communicate. This includes factors like whether links are directed or undirected, static or time-varying. The State-of-the-art algorithms and their analyses are tailored to these different scenarios, highlighting the crucial role of network topology [2]. It has also been highlighted that denser networks tend to accelerate convergence due to faster information propagation but at the cost of increased communication overhead, whereas sparser networks reduce communication costs but may require more iterations to reach consensus.

In our research, our primary focus is in the effects of quantization, erasure and network topology on communication efficiency and convergence in distributed optimization. By understanding how these factors interact, we aim to provide insights into how quantization errors and message loss affect the performance of distributed optimization algorithms.

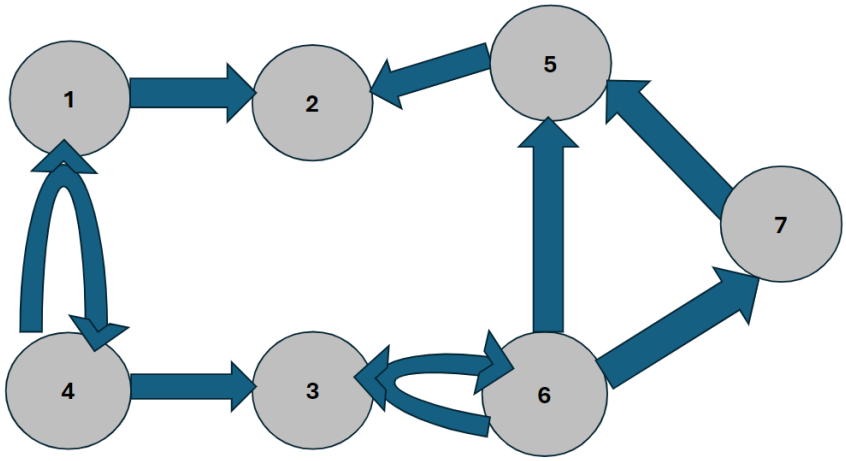

Figure 1.1: Communication grapgh for a distributed optimatation

1.1 Background

Over the past few years, distributed optimization and distributed machine learning (DML) have become increasingly important, particularly due to the rise of large-scale data processing and the development of highly distributed computing infrastructures. Distributed approaches to optimization aim to breaks the complex problems into smaller, more manageable subproblems that can be solved collaboratively by multiple agents or nodes in a network. These methods are crucial for

applications where data is inherently distributed (e.g., edge computing, federated learning, and cloud-based systems) or when data cannot be centralized due to privacy or resource constraints. This section provides an overview of centralized and decentralized methods in distributed optimization, highlighting key techniques like gossip-based and dual-based methods.

Centralized optimization methods rely on a central controller that collects information from all nodes, solves a global optimization problem, and disseminates the solution back to the nodes. Methods like Stochastic Gradient Descent (SGD) and Gradient Boosting Machines exemplify this centralized approach, requiring a central server to aggregate gradients or updates after each iteration. While these methods can achieve high accuracy and convergence, they suffer from several drawbacks in distributed settings. Specifically, centralized methods are sensitive to single points of failure, are limited by the capacity of the central server, and can be inefficient due to the need for massive data transfer. To address these limitations, recent advancements have focused on distributed optimization techniques. Downpour SGD is introduced as a variant of asynchronous stochastic gradient descent designed to handle large data sets by using multiple replicas of a single DistBelief model[3]. This method, along with Sandblaster L-BFGS, leverages distributed systems to mitigate the drawbacks of centralized methods. By distributing the computational load across multiple nodes and reducing dependence on a single central server, these distributed approaches significantly improve efficiency and scalability.

Federated optimization refers to the optimization problem implicit in federated learning, drawing a connection (and contrast) to distributed optimization. Federated optimization has several significant characteristics that distinguish it from a conventional distributed optimization problem: (1)Non-IID: The training data on a given client is typically based on the usage of the mobile device by a particular user, and hence any particular users local dataset will not be representative of the population distribution. (2)Unbalanced: Similarly, some users will make much heavier use of the service or app than others, leading to varying amounts of local training data. (3)Massively distributed: The number of clients participating in an optimization to be much larger than the average number of examples per client. (4)Limited communication: Mobile devices are frequently offline or on slow or expensive connections[4].

Decentralized optimization techniques distribute the computation workload across nodes, each solving a local problem and sharing information with its immediate neighbors. This approach removes the dependency on a central node, enhancing robustness and scalability. However, this methods faces challenges such as slower convergence due to limited information sharing and the need for consensus across nodes. Decentralized methods are often seen in environments such as wireless sensor networks, edge computing. where data privacy and communication overhead are critical concerns.

In recent years, Federated Learning (FL) has emerged as a groundbreaking approach in distributed machine learning, particularly suited to the challenges posed by sensi-

tive and voluminous data generated by mobile and edge devices[5], [6]. FL enables model training directly on devices, preserving data privacy and reducing the need for central data aggregation[7]. A pivotal algorithm in Federated Learning is Federated Averaging (FedAvg), which combines local stochastic gradient descent with server-side model averaging. FedAvg is notably effective in handling non-IID (Independent and Identically Distributed) data and reducing communication overhead compared to traditional methods[7], [8].

Consensus methods are another class of decentralized approaches where nodes iteratively communicate with their neighbors to reach agreement on the global solution. Methods such as Distributed Consensus-ADMM and Gradient Descent (DGD) belong to this category, ensuring that all nodes converge to a common solution over time. Nedic et al.[9] considered several algorithms that use different types of consensus models, namely weighted-averaging and push-sum models[10].

Alternating direction method of multipliers (ADMM)[11] is a simple but powerful algorithm that is well suited to distributed convex optimization, and in particular to problems arising in applied statistics and machine learning. It takes the form of a decomposition-coordination procedure, in which the solutions to small local subproblems are coordinated to find a solution to a large global problem. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization, two earlier approaches. It turns out to be equivalent or closely related to many other algorithms as well, such as Douglas-Rachford splitting from numerical analysis, Spingarns method of partial inverses, Dykstras alternating projections method, Bregman iterative algorithms for problems in signal processing, proximal methods, and many others. The fact that it has been re-invented in different fields over the decades underscores the intuitive appeal of the approach.

Gossip-based methods also known as gossip algorithm are motivated by applications to sensor, peer-to-peer, and ad hoc networks for exchanging information and computing in an arbitrarily connected network of nodes. These constraints motivate the design of simple decentralized algorithms for computation where each node exchanges information with only a few of its immediate neighbors in a time instance (or, a round). The goal in this setting is to design algorithms so that the desired computation and communication are done as quickly and efficiently as possible. The averaging time of a gossip algorithm depends on the second largest eigenvalue of a doubly stochastic matrix characterizing the algorithm. Designing the fastest gossip algorithm corresponds to minimizing this eigenvalue, which is a semidefinite program (SDP). There two well-known cases for which the performance and scaling of gossip matrices are studied: Wireless Sensor Networks, which are modeled as Geometric Random Graphs, and the Internet graph under the so-called Preferential Connectivity (PC) model[12].

A key focus of recent research is improving distributed optimization methods. The paper titled "Push-Pull Gradient Methods for Distributed Optimization in Networks"

by Shi Pu, Wei Shi, Jinming Xu, and Angelia Nedi introduces a novel gradient-based algorithm for distributed (consensus-based) optimization in directed graphs. This new approach differs from previous push-sum protocols[13], [14] by using a row stochastic matrix to mix decision variables and a column stochastic matrix to track average gradients. The paper[15] also explores a random-gossip variant of the push-pull method, known as the G-Push-Pull algorithm. In this variant, agents communicate randomly with one or two of their neighbors during each iteration. The authors show that both the Push-Pull and G-Push-Pull methods converge linearly to the optimal solution for strongly convex and smooth objective functions. Ongoing research aims to refine these methods and enhance communication-efficient distributed learning techniques.

1.2 Previous Work on FEDAVG and PUSHPULL algorithms

The field of decentralized and distributed optimization has seen substantial progress, particularly in consensus-based methods over both undirected and directed graphs. Early research efforts focused on static undirected networks and leveraged algorithms like the Alternating Direction Method of Multipliers (ADMM), achieving linear convergence through the use of doubly stochastic matrices for consensus [16][17]. However, the construction of doubly stochastic matrices can be impractical for directed graphs, leading to the development of push-sum algorithms that facilitate consensus in directed networks[18], [19]. These methods were later extended to accommodate time-varying graphs and smooth, strongly convex functions [15], [20], [21], although they often required careful step size selection, which posed stability challenges that have been addressed in more recent work[13].

The Federated Averaging (FedAvg) approach, developed by [4], [7],combines local stochastic gradient descent with model averaging at the server to handle federated learning difficulties such as non-IID data and communication restrictions. This improves on past distributed optimisation efforts, such as those by [22] and [23], which focused on cluster-based settings but did not address the particular difficulties of federated settings. FedAvg is contrasted to classic approaches such as synchronous SGD[8] and asynchronous SGD[3], demonstrating its effectiveness in lowering communication overhead. It also contains privacy considerations, relying on secure multiparty computing and differential privacy[24] to alleviate vulnerabilities associated with on-device data.

In particular, the push-pull gradient methods introduced in these works provide a significant foundation for decentralized optimization, where row-stochastic matrices are used for decision variable updates and column-stochastic matrices for gradient tracking. These methods unify different network architectures, including peer-to-peer, master-slave, and leader-follower systems, allowing for flexibility in distributed optimization environments. Notably, these algorithms have also been extended to handle asynchronous communication through random-gossip variants, making them

robust to network imperfections such as delays and stochastic updates[25][26].

1.3 Goals

This study evaluates of Federated Averaging (FEDAVG) and push-pull optimization algorithms within the context of distributed machine learning, particularly in high-performance computing (HPC) environments using MPI (Message Passing Interface) communication protocols. The study will examine how these algorithms perform under various communication constraints and predefined network topologies, such as fully connected, circular, star-shaped, and random networks, involving interconnected nodes within an HPC cluster.

The primary objective is to explore the impact of communication imperfections on important algorithm metrics, specifically the Quantization, used to reduce communication overhead by approximating continuous values with fewer bits, can introduce errors (quantization errors)and erasure flaws that affect the precision of model updates, potentially slowing convergence and reducing accuracy. The study will specifically focus on FEDAVG and push-pull optimization algorithms, intentionally excluding other decentralized machine learning techniques from the scope of analysis.

Experiments are conducted within an HPC cluster environment, utilizing MPI for communication between nodes. This setup allows for the study of algorithm performance under realistic conditions, reflecting the challenges encountered in practical distributed machine learning scenarios. The investigation will be limited to predetermined network topologies and specific communication techniques, with the purpose of understanding and documenting the effects of communication defects on algorithm performance under different scenarios.

This delimitation ensures that the research remains focused on its core objectives, providing detailed insights into the specific challenges and performance characteristics of FEDAVG and push-pull algorithms in the context of quantization, erasure effects, and network topology variations within an HPC framework.

This dissertation methodically investigates federated learning algorithms, starting with implementing two core techniques: using FedAvg in a star topology and a push-pull mechanism in a peer-to-peer network. These applications act as standard models, setting a starting point for additional examination. The study initially presents quantization methods to FedAvg and push-pull algorithms, carefully analyzing their impact on communication costs, model performance, and convergence. Quantization aims to improve communication efficiency by decreasing data precision, and this research assesses the compromises associated with this strategy. After this, the thesis explores how resilient these algorithms are when messages is deleted. Understanding the impact of message loss on federated learning's robustness and reliability is essential for gaining insights into the fault tolerance of the algorithms.

The study then moves on to examine how various communication topologies impact

the effectiveness of federated learning by investigating the role of network topology. By testing different topological designs, the research identifies configurations that optimize both training speed and accuracy, revealing insights into how network topology influences the results of federated learning.

In the last stage, the thesis evaluates how well the baseline algorithms FedAvg and push-pullcan scale with varying network sizes, using the MNIST and CIFAR-10 datasets. This analysis of scalability plays a crucial role in assessing the algorithms' suitability for widespread implementation in practical situations. In the thesis, there is a constant focus on the relationship between communication efficiency, accuracy, and convergence, offering a thorough insight into the performance variations in federated learning under various circumstances.

2

Theory

2.1 Distributed Optimization

Distributed optimization is an important technique used to solve large scale machine learning problems where data is distributed across multiple nodes/clients that communicate over network. Each agent (i) in a network aims to minimize a its local objective function ($f_i(x)$), while sharing information with each other to ensure convergence to the optimal solution, which is usually the sum of local objectives from all node. Communication between nodes is sometimes limited by bandwidth or latency constraints, necessitating the optimization of both computation and communication.

Each node i has its own local data and aims to minimize a local objective function $f_i()x$. The main goal of distributed systems is to find the value of x that minimizes the sum of all the local objective functions from every node, represented as:

$$\min_x \sum_{i=1}^{N} f_i(x)$$

Where:

- N is the number of nodes (or clients).
- $f_i(x)$ is the local objective function at node i, which depends on local data.
- Each node/clients computes its gradient $\nabla f_i(x)$ locally, and the clinets communicate with each other or a central server to update x in a coordinated manner.

The nodes/clients communicate over a communication graph. The communication graph (G) represent how nodes communicate each other and structure of the communication graph has considerable impacts the performance of the distributed optimization process. A well-connected graph can helps speed up information transmission and convergence. Conversely, a poorly connected graph can slow down the process and lead to poor performance [27].

The communication graph $G = (V, E)$ plays an important role in determining how fast and efficiently consensus can be achieved, where V represents the nodes and E the communication links between nodes. Each node in the graph represents an agent, and each edge represents a communication link between two agents. The graph can be directed or undirected

In symmetric communication[28], data flow between nodes is bidirectional, which

means that node A can transmit information to node B and node B can send information back to node A. This sort of communication is widespread in balanced and evenly connected networks, resulting in easier analysis and faster consensus.

Asymmetric communication [28] occurs when information transmission is one-way, i.e., node A can send data to node B but node B cannot send data back. Asymmetric communication can occur due to limitations in network resources or hierarchical systems. While it complicates analysis and may hinder consensus, it can simulate actual circumstances like master-slave or client-server relationships. Symmetric communication simplifies convergence analysis and frequently leads to speedier consensus across nodes, although asymmetric communication, while more complex, can represent practical systems with limited data flow.

The communication graph is central to the distributed consensus algorithm, the consensus algorithm plays an important role to distributed and multi-agent systems. Its goal is to ensure that all nodes (or agents) in a system agree on the same data value, even if some nodes fail or deliver inaccurate data [29]. A consensus algorithm ensures that each node i updates its local decision variable x_i by incorporating information from its neighboring nodes j. Nodes in a network uses communication graphs to repeatedly update their local decision variables by incorporating information from neighboring nodes, thus achieving both local convergence and global consensus [29]. The consensus update rule can be represented as:

$$x_i^{t+1} = x_i^t - \eta \sum_{j \in \mathcal{N}_i} W_{ij}(x_i^t - x_j^t)$$

where:

- η is the learning rate,
- W_{ij} represents the communication weight between node i and node j,
- $\mathcal{N}_i$ is the set of neighbors of node i.

2.2 Overview of Algorithms

2.2.1 FEDAVG

Federated Averaging (FEDAVG) is a widely used algorithm in Federated Learning, where multiple devices (clients) collaborate to train a single, powerful model without sharing their private data. Each client trains a copy of the model using its local data and after completion of the local training, the client does not sends this data to server instead, it sends only the updates model parameters (gradients or weights) to a central server. The server averages these parameters from all devices, this averaging step is core of FedAvg algorithm and broadcasts the averaged parameters back to the clients. Each devices receive the averaged parameter and update its local model copy accordingly. This process of local training,sending updates, averaging them and updating the global model is repeated over several rounds, gradually improving the global model[30][4]. Traditionally FedAvg uses central server to averaging the

updates, but it can be adapted to decentralized topologies (FC or circular).

Federated Learning is a machine learning approach for training models on clients without sharing original data, for Next-Word Prediction on Mobile Phones. The diagram 2.6 illustrates a distributed version of federated learning, where remote devices/clients communicate directly with each other without need for central server. Each client performs local training on its own data and then exchanges model updates with other devices in a peer-to-peer fashion, shown by the dashed communication lines between smartphones [31]. The privacy of text data is maintained by training a prediction model locally on non-identically dispersed user data and delivering updates to the server. The server aggregates these updates to form a new global model, iteratively improving predictions until convergence or a stopping criterion is met.This decentralized/distributed approach reduces reliance on a central server and allows devices to collaborate more flexibly, although it introduces additional complexity in communication and coordination.

Both 2.1 and 2.2 illustrate two important aspects of Federated Learning (FL), showcasing both the traditional centralized and decentralized approaches respectively.

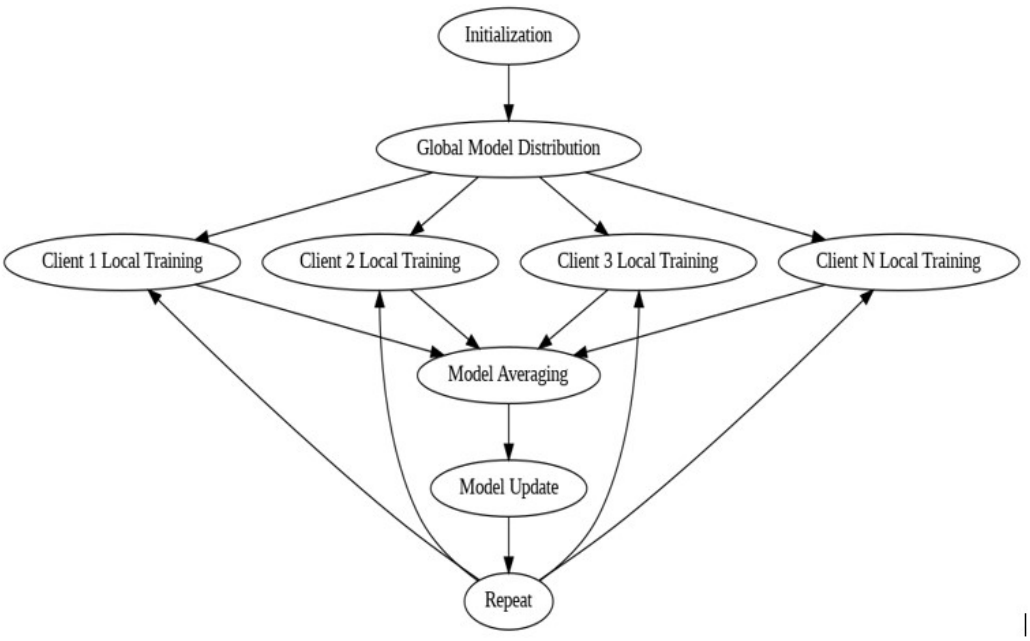

Figure 2.1: Centralized Federated Averaging

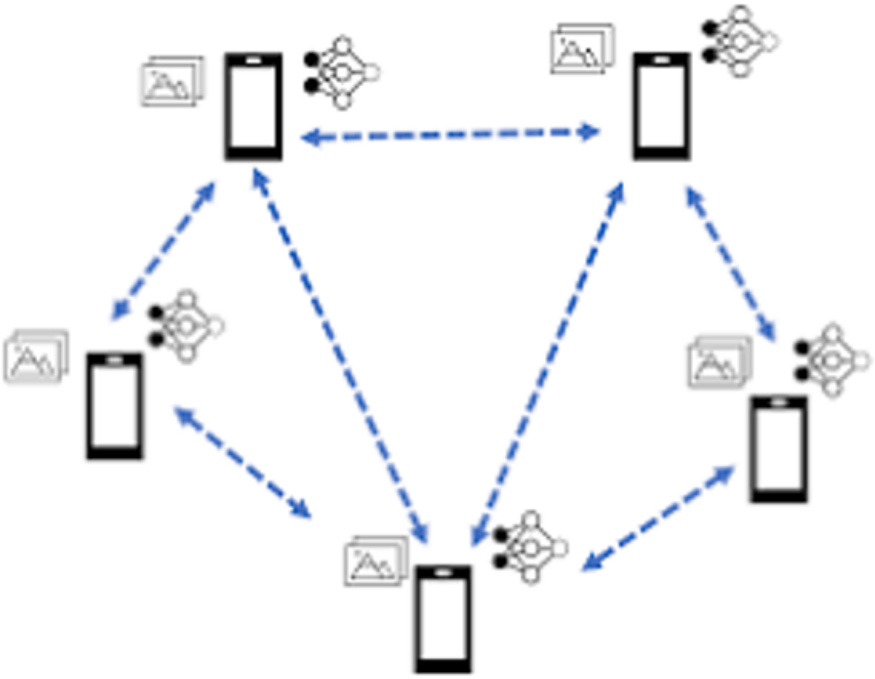

Figure 2.2: Decentralized Federated Averaging [31]

Constantly updating the model parameters on a regular basis, which for deep neural network can involve large number of parameters, requires frequent transmission of substantial amounts of data between cilents and server. This creates a communication overhead between central server and clients/devices, especially in large-scale-networks with many participants, which can significantly slow down the model's convergence. Given that communication channels between devices/clients and the server usually have limited bandwidth and power, this frequent data transfer becomes a bottleneck.To address this, techniques such as quantization, compression, and adaptive communication strategies are often employed to reduce the communication load while maintaining model performance.[32][30]. However, in our study we are concentrating on the quantization techniques, how its useful in saving communication time and also its effect on convergence of the algorithm.

2.2.1.1 Algorithm: FedAvg

In the FedAvg algorithm, D is entire dataset used for training and is distributed among clients, $w(0)$ is inital global model,which is same for all clients at begining of training process. Learning rate η and $w(T)$ is updated global model after specified number of training rounds.FedAvg algorithm involves two steps one local updates and global updates .
In local update each client calculates average gradient (∇L_i) over its own dataset D_i and client updates its local model $w(t)_i$.

$$w(t)_i = w(t-1)_i - \eta \cdot \nabla L_i$$

Where, $w(t-1)_i$ is clients local model from previous round (t-1), η is learning rate, ∇L_i is average gradient calculated over client's local data.

Data: $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, Global Model $w^{(0)}$, Learning Rate η

Result: Updated Global Model $w^{(T)}$

for *each round* $t = 1, 2, \ldots, T$ **do**

  for *each client* $i = 1, 2, \ldots, N$ **do**

    Calculate the average gradient over dataset $\mathcal{D}_i$: $\bar{\nabla}\mathcal{L}_i = \frac{1}{|\mathcal{D}_i|} \sum_{x \in \mathcal{D}_i} \nabla \mathcal{L}(w^{(t-1)}; x)$

    Update local model: $w_i^{(t)} = w^{(t-1)} - \eta \cdot \bar{\nabla}\mathcal{L}_i$

  Aggregate local updates: $w^{(t)} = \sum_{i=1}^{N} p_i w_i^{(t)}$

Figure 2.3: Algorithm:FedAvg [33]

In global update, server aggregates the local model updates from all clients.

$$w(t) = \frac{\sum_{i=1}^{N}(p_i \cdot w(t)_i)}{\sum_{i=1}^{N} p_i}$$

Where, $w(t)$ is updated model, p_i is weight of client i, $w(t)_i$ is local update from client i.
FedAvg algorithm iterates through these two steps for specified number of rounds (T),gradually improving the local and global models based on collabrative learninig from all clients dataset [33].

2.2.2 PUSH-PULL

Push-Pull Optimization: Push-Pull algorithms are essential tools for distributed optimization, enabling collaborative optimization among interconnected agents or nodes in various network topologies. These algorithms use local computations to reach consensus on a global objective function, facilitating effective information exchange and synchronization among agents.

Now, we explain the Push-Pull Gradient Method, introduced by Author(s) Shi Pu, Wei Shi, Jinming Xu, and Angelia Nedic. Push-Pull Gradient Methods for Distributed Optimization in Networks [34], unifies different types of distributed architectures, including decentralized, centralized, and semi-centralized architectures. This method provides a cohesive framework that bridges various distributed paradigms, offering a unified approach to optimizing computational processes across diverse system configurations.If the graph is directed and strongly connected, For the fully

decentralized case, suppose we have a graph G that is undirected and connected. Here, G represents the network topology. We can set $G_R = G_C = G$, where G_R and G_C are the adjacency matrices for the receiver and controller nodes, respectively. In this context, R and C are symmetric matrices representing specific weight assignments within the network, and the proposed algorithm reduces to the one considered in [13], [35]. If the graph is directed and strongly connected, we can still let $G_R = G_C = G$ and design the weights for R and C accordingly.

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0.5 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0.5 & 0.5 & 0.5 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}.$$

The matrix R defines the weight distribution for the network when information is pulled by the neighbors (receiver nodes). The first row has a weight of 1 for the first node and 0 for others, indicating that node 1's information is solely determined by itself. The other rows distribute the information equally between two nodes, reflecting the connections in a decentralized or partially centralized system where nodes other than node 1 have some degree of interaction with multiple neighbors.

The matrix C defines the weight distribution when information is pushed to the neighbors (controller nodes). The first row reflects that node 1 pushes its information equally to all other nodes, consistent with its role as a central node in a semi-centralized architecture. The other rows have lower weights, reflecting a less active role in the push process for nodes 2, 3, and 4, which mainly respond to node 1.

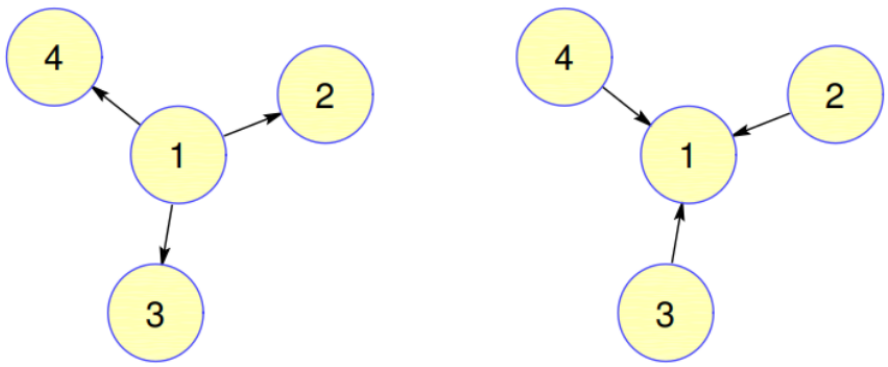

Figure 2.4: illustrates the Push-Pull Algorithm working mechanism. The left side of the figure shows node 1 actively pulling information from nodes 2, 3, and 4, while the right side shows node 1 pushing information to nodes 2, 3, and 4. This visualizes how node 1 acts as a central hub, pulling information in one phase and pushing information in the other phase, consistent with the push-pull dynamics[34].

To illustrate the less straightforward situation of (semi-)centralized networks, let us provide a simple example. Consider a four-node star network composed of nodes 1,

2, 3, and 4, where node 1 is situated at the center and nodes 2, 3, and 4 are bidirectionally connected with node 1 but not connected to each other. In this case, the matrix R in our algorithm can be chosen as 1s information regarding x_{1k} is pulled by the neighbors (the entire network in this case) through G_R; the others only passively infuse the information from node 1. At the same time, node 1 has been pushed information regarding y_{ik} (for $i = 2, 3, 4$) from the neighbors through G_C; the other nodes only actively comply with the request from node 1. This motivates the algorithms name, *push-pull gradient method*. Although nodes 2, 3, and 4 are updating their y_i values accordingly, these quantities do not have to contribute to the optimization procedure and will die out geometrically fast due to the weights in the last three rows of C. Consequently, in this special case, the local step size for agents 2, 3, and 4 can be set to 0. Without loss of generality, suppose $f_1(x) = 0$. Then the algorithm becomes a typical centralized algorithm for minimizing $\sum_{i=2}^{4} f_i(x)$ where the master node 1 utilizes the slave nodes 2, 3, and 4 to compute the gradient information in a distributed way. Taking the above as an example for explaining the semi-centralized case, it is worth noting that node 1 can be replaced by a strongly connected subnet in G_R and G_C, respectively. Correspondingly, nodes 2, 3, and 4 can all be replaced by subnets as long as the information from the master layer in these subnets can be diffused to all the slave layer agents in G_R, while the information from all the slave layer agents can be diffused to the master layer in G_C. Specific requirements on the connectivities of slave subnets can be understood by using the concept of rooted trees. We refer to the nodes as leaders if their roles in the network are similar to the role of node 1; and the other nodes are termed as followers. Note that after the replacement of the individual nodes by subnets, the network structure in all subnets is decentralized, while the relationship between the leader subnet and follower subnets is master-slave. This is why we refer to such an architecture as semi-centralized. This example illustrates a semi-centralized case where node 1 can be replaced by a strongly connected subnet in G_R and G_C respectively. Similarly, nodes 2, 3, and 4 can be replaced by subnets, ensuring that information can be diffused appropriately between master and slave layers in G_R and G_C. The connectivity requirements of these slave subnets can be understood using rooted trees. Nodes fulfilling roles similar to node 1 are termed leaders, while other nodes are termed followers based on their network roles [34].

Figure 2.5: Push-pull block diagram

Sender Node A -> Receiver and Requester Node B: The arrow from Node A to Node B is labelled "Push Request" suggesting that Node A is passing data to Node B without receiving a request.

Receiver Node B -> Responder Node C: The arrow from Node B to Node C is labelled "Pull Request" indicating that Node B is seeking data from Node C.
Responder Node C -> Receiver, Requester Node B: The arrow from Node C to Node B is labelled "Response" indicating that Node C is returning the requested data to Node B.
Where nodes both push data to specific recipients and respond to pull requests from other nodes.From the viewpoint of an agent, the information about the gradients is pushed to the neighbors, while the information about the decision variable is pulled from the neighbors hence giving the name push-pull gradient methods[34].

The 2.5 diagram provide a visual representation of how push and pull algorithms work in distributed systems, where nodes exchange data or messages based on different communication patterns.
In a push phase of distributed optimization, each node computes its local updates based on its own data and current model parameters. These local updates are then transmitted or "pushed" to neighboring nodes in the network. This dissemination of information enables nodes to share their insights and collaborate effectively towards a common objective.

Conversely, in the pull phase, nodes receive updates or information from their neighbors. This process involves aggregating the received updates, typically through techniques like averaging or consensus, to update their own local model parameters. The pull phase ensures that nodes benefit from the collective intelligence of neighboring nodes to enhance their own understanding and refine their model parameters.

However, in practical scenarios with imperfect communication channels, such as quantization or erasure channels, the efficacy of push and pull algorithms can be challenged. Erasure channels may drop transmitted data packets. These imperfections can lead to loss of information or introduce errors in the communication process.

2.2.2.1 Algorithm: Push-Pull

Algorithm 1 (pushpull) can be rewritten in the following aggregated form.

$$\mathbf{x}_{k+1} = \mathbf{R}\left(\mathbf{x}_k - \boldsymbol{\alpha}\mathbf{y}_k\right) \tag{2a}$$

$$\mathbf{y}_{k+1} = \mathbf{C}\mathbf{y}_k + \nabla F\left(\mathbf{x}_{k+1}\right) - \nabla F\left(\mathbf{x}_k\right) \tag{2b}$$

A matrix is nonnegative if all its elements are nonnegative.
In the context of the algorithm, let $\alpha = \text{diag}\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a nonnegative diagonal matrix, and $R = [R_{ij}], C = [C_{ij}] \in \mathbb{R}^{n\times n}$. We make the following assumption regarding the matrices R and C:
Assumption 2: The matrix $R \in \mathbb{R}^{n\times n}$ is a nonnegative row-stochastic matrix, and $C \in \mathbb{R}^{n\times n}$ is a nonnegative column-stochastic matrix. This means that R

Algorithm 1: Push–Pull.

Each agent i chooses its local step size $\alpha_i \geq 0$,
in-bound mixing/pulling weights $R_{ij} \geq 0$ for all $j \in \mathcal{N}^{\text{in}}_{\mathbf{R},i}$,
and out-bound pushing weights $C_{li} \geq 0$ for all $l \in \mathcal{N}^{\text{out}}_{\mathbf{C},i}$;
Each agent i initializes with any arbitrary $x_{i,0} \in \mathbb{R}^p$ and $y_{i,0} = \nabla f_i(x_{i,0})$;
for $k = 0, 1, \ldots,$ **do**
  for each $i \in \mathcal{N}$,
  agent i pulls $(x_{j,k} - \alpha_j y_{j,k})$ from each $j \in \mathcal{N}^{\text{in}}_{\mathbf{R},i}$;
  agent i pushes $C_{li} y_{i,k}$ to each $l \in \mathcal{N}^{\text{out}}_{\mathbf{C},i}$;
  for each $i \in \mathcal{N}$,
    $x_{i,k+1} = \sum_{j=1}^{n} R_{ij}(x_{j,k} - \alpha_j y_{j,k})$;
    $y_{i,k+1} = \sum_{j=1}^{n} C_{ij} y_{j,k} + \nabla f_i(x_{i,k+1}) - \nabla f_i(x_{i,k})$;
end for

Figure 2.6: Algorithm:Push-pull[34]

satisfies $R\mathbf{1} = \mathbf{1}$, and C satisfies $\mathbf{1}^T C = \mathbf{1}^T$. Additionally, the diagonal elements of both R and C are positive, i.e., $R_{ii} > 0$ and $C_{ii} > 0$ for all $i \in \mathcal{N}$.
As a consequence of C being column-stochastic, it can be shown by induction that:

$$\frac{1}{n}\mathbf{1}^T y_k = \frac{1}{n}\mathbf{1}^T \nabla F(x_k), \quad \forall k.$$

This relation is crucial as it allows (a subset of) the agents to track the average gradient $\frac{1}{n}\mathbf{1}^T \nabla F(x_k)$ through the y-update.
In each iteration, every agent will share gradient information with its outgoing neighbors and receive decision variables from its incoming neighbors. Each component of y_k tracks average gradients using the column-stochastic matrix C, while each component of x_k drives optimization through average consensus using the row-stochastic matrix R. Algorithm 1 resembles gradient tracking methods seen in prior work, where the doubly stochastic matrix is split into row-stochastic and column-stochastic components. This asymmetric $R-C$ structure, previously used for achieving average consensus, differs from linear systems due to its introduction of nonlinear dynamics from the gradient terms.
Now, we establish the graph structures G_R and G_C induced by matrices R and C respectively. Notably, G_C mirrors G_C with all its edges reversed[34].

2.2.3 Evaluation Metrics

In the field of federated learning and distributed machine learning, the performance and efficiency of algorithms are critically influenced by several key factors. Understanding these factors is essential for optimizing algorithm performance and ensuring robust, scalable machine learning models.

In our project, we implemented and analyzed key metrics such as communication time, average loss, convergence threshold, convergence time, convergence loop, and accuracy.

Average Communication Time: Communication time measure shows how long it takes a client to exchange gradients with all other clients that are participating in the training. It includes both the time to receive gradients synchronously from all clients plus the time to send its gradients synchronously to all other clients. Accurate measurement of communication time is essential for understanding the efficiency of distributed training processes.The Average Communication Time represents the average time spent by all clients during each training iteration to exchange gradients in a distributed setup.

The accumulated total communication time over multiple iterations is given by:

$$\text{Total Comm Time} = \sum_{i=1}^{N} T_{\text{comm},i}$$

Where:

- N = Total number of iterations.
- $T_{\text{comm},i}$ = Communication time during the i-th iteration, which includes both the reduction and broadcast times.

The Average Communication Time is calculated as:

$$\text{Average Comm Time} = \frac{\text{Total Comm Time}}{\text{Number of Clients}}$$

Where:

- Total Comm Time is the sum of communication times from all clients.
- Number of Clients is the total number of clients participating in the training process.

This formula effectively represents the overall communication overhead experienced during the training process.

Convergence Time: Convergence time consumed by a client from the start of training until it reaches a point where the training loss falls below a predefined convergence threshold (0.01 in our experiment). This metric, typically measured in seconds, reflect the speed at which the model stabilizes and begins producing reliable outputs. Factors such as hardware specifications, dataset size, and model complexity can influence convergence time. The Average Convergence Time is the average time taken by all clients to reach convergence, where the training loss falls below a predefined threshold.

The convergence threshold ϵ is a predefined value that determines when the training is considered to have converged. In our experiment, this threshold is set to:

$$\epsilon = 0.01$$

The training is considered to have converged when the average loss falls below this threshold. The individual client convergence time

$$T_{\text{convergence}} = T_k - T_{\text{start}} \quad \text{where} \quad L_{\text{avg},k} < \epsilon$$

Step-by-step Representation:

1. Start timing at T_{start}.
2. For each epoch k, compute the average loss $L_{\text{avg},k}$.
3. Identify the first epoch k where (average loss) $L_{\text{avg},k} < \epsilon$ (threshold)
4. Calculate the convergence time as the difference between the end time of this epoch and the start time.

The average loss $L_{\text{avg},k}$ for the k-th epoch is calculated as the mean of the loss values over all batches within that epoch. It measures how well the model is learning during the training process.

$$L_{\text{avg},k} = \frac{1}{n} \sum_{i=1}^{n} L_{i,k}$$

Where:

- n = Total number of batches in the k-th epoch (`len(train_loader)`).
- $L_{i,k}$ = Loss value of the i-th batch in the k-th epoch.

Total Convergence time across all clients that have reached convergence is given by.

$$T_{\text{convergence}} = \sum_{j=1}^{M} T_{\text{convergence},j}$$

Where:

- M = Total number of clients that have reached convergence.
- $T_{\text{convergence},j}$ = Convergence time for the j-th client.

The Average Convergence Time is calculated as:

$$\text{Average Convergence Time} = \frac{T_{\text{convergence}}}{\text{Number of Clients}}$$

Where:

- $T_{\text{convergence}}$ is the sum of convergence times from all clients who have reached convergence.
- Number of Clients is the total number of clients participating in the training process.

Convergence Loops: This metric refers to the specific training loop during which the training loss drops below the predefined convergence threshold (in this case convergence threshold of 0.01). The number of loops required provides insights into the stability and efficiency of the training process, indicating how quickly the model is able to meet the convergence criteria.

Accuracy: In this context, accuracy refers to the test accuracy, which measures how well the trained model performs on a separate test dataset that was not used during

training. It is a key indicator of the model's ability to generalize to new, unseen data reflecting the effectiveness of the model in practical applications.

These metrics communication time, convergence time, convergence loops, and accuracy were specifically measured during the implementation of distributed optimization algorithms, including the FedAvg and Push-Pull. By defining, implementing, and analyzing these measures within our code, we gained valuable insights into the real-world difficulties and trade-offs associated with distributed learning settings.
In this study, we focused on implementation of federated averaging algorithm and push-pull algorithm with specific attention to the data partitioning strategies, quantization of gradients and erasure of gradients. We also emphasized to analyze the impact of different network topologies with these metrics.

2.3 Quantization

Quantization is a process of mapping continuous values to discrete levels, commonly used in machine learning to reduce the size of models and the volume of data transmitted during training in distributed systems. We can apply quantization in two ways Post-training quantization and quantization aware training. In post-training quantization, quantizes models or gradients after training has been completed. In the case of gradient quantization, this would mean applying quantization techniques to the computed gradients after the full-precision training has been completed. This is useful in scenario where the goal is to minimize the communication overhead. Whereas in quantization aware training, quantization is integrated during training phase, leading to better-optimized models with minimal accuracy degradation, but it requires more computational resources and time due to the need for retraining [36].

Gradient quantization addresses the problem of bandwidth and communication cost, which are significant bottlenecks in decentralized and distributed learning systems. In gradient quantization [37], gradients are scaled to a range determined by the number of bits and then rounded to the nearest quantization level.
This quantization step maps each gradient component to a discrete level, reducing its precision but significantly compressing the amount of data transmitted.

Let:

- $g \in \mathbb{R}^d$ be the original gradient vector (tensor).
- $\text{max\_val} = \max(|g|)$ be the maximum absolute value of the gradient.
- b be the number of bits used for quantization.

The quantization process can be broken down into normalization, scaling, rounding, and clamping.
The gradient is normalized using the maximum absolute value:

$$\hat{g}_i = \frac{g_i}{\text{max\_val}}$$

where $\hat{g}_i$ is the normalized version of the gradient.
The number of discrete levels available for quantization is determined by b bits. The

range of quantization levels is:

$$\text{scale} = 2^{b-1} - 1$$

The normalized gradient is scaled to this range:

$$\tilde{g}_i = \hat{g}_i \times \text{scale}$$

where $\tilde{g}_i$ is the scaled gradient ready for quantization. The scaled gradient is rounded to the nearest integer to map it to a discrete level:

$$q(g_i) = \text{round}(\tilde{g}_i)$$

The rounded value is clamped to ensure it lies within the range $[-\text{scale}, \text{scale}]$:

$$q(g_i) = \min(\max(q(g_i), -\text{scale}), \text{scale})$$

where $q(g_i)$ is the quantized value of g_i.

For Dequantization of multi-bit case ,

$$\hat{g}_i = \frac{q(g_i)}{2^{b-1} - 1} \times \text{max\_val}$$

For the special case where $b = 1$, the gradient [38] is simply quantized to its sign :

$$q(g_i) = \text{sign}(g_i)$$

For Dequantization $b = 1$,

$$\hat{g}_i = q(g_i) \times \text{max\_val}$$

where:

$$\text{sign}(g_i) = \begin{cases} +1, & \text{if } g_i > 0 \\ -1, & \text{if } g_i < 0 \\ 0, & \text{if } g_i = 0 \end{cases}$$

Quantization also introduces a loss of precision by representing weights or gradients with fewer bits. while this reduces communication overhead however this reduces precision may introduce small errors called quantization error which affects accuracy of the model, convergence, and number of iteration needed for convergence compared to a non-quantized model.

2.3.1 Trade-offs in Quantization

Optimizing quantization requires balancing between communication efficiency, computational complexity, and model accuracy. The selection of an optimal number of quantization levels/bits is critical in navigating this trade-off. Reducing quantization levels/bits minimizes communication overhead but it introduces the risk

of quantization error, leading to a loss of precision and slower convergence. Conversely, increasing more quantization levels/bits enhances data precision but also raises computational cost and communication time. The challenge lies in determining the optimal quantization level based on specific distributed learning application requirements.

2.3.1.1 Reduced Model Size and Communication time

Using fewer bits of quantization levels reduces the size of the model updates, decreasing the amount of data transmitted between clients and server.This also reduce communication overhead as less data needs to be communicated between clients and server. since smaller data packets can be transmitted more quickly and also model size requires less memory on device . However, this reduction can lead to more distortion and potentially lower model accuracy due to information loss.[39][40]

2.3.1.2 Communication Efficiency and Computational Overhead

Quantization may offer communication efficiency but in turn it increases computation overhead, as lowering the model parameter and gradients from the original format to lower bit forms and back again de-quantization may add extra computations [41].
Quantization introduces information loss. This quantized values may not be as exact as the original numbers, depending on bit width that was used.This may result in more calculations to make up for decreased precision. And slow convergence due to quantization error. Finding the right balance between these factors is crucial for optimizing the performance of distributed systems, especially in scenarios involving resource-constrained devices where efficient computation is vital for successful model training and convergence.

2.3.1.3 Optimal balance in Quantization

Optimizing quantization requires a careful approach to balance between factors like, communication efficiency, computational complexity, and data precision in our project's distributed learning application.
Employing more quantization levels improves data precision but comes at the cost of increased computational complexity and communication overhead as discussed in above section. The challenge lies in determining the optimal quantization level tailored to our specific distributed learning application requirements. This involves considering the trade-offs between communication efficiency, data precision, and computational resources.
By navigating these trade-offs effectively , we can optimize the performance and efficiency of our distributed learning application while maintaining communication reliability and computation efficiency and data precision.

2.3.2 Quantization in FedAvg and push-pull

In a federated learning, the gradients computed locally on each device are sent back to a central server for averaging. However, transmitting high-precision gradients from millions of devices/clients results in communication overhead. The use of gradient quantization reduces this overhead by compressing the gradients, thereby allowing the learning process to be more communication-efficient[4].

In order to alleviate the limitation of bandwidth in distributed networks, quantized messages should be transmitted between the nodes. To solve this distributed optimization problem, a gradient descent method is combined with a distributed quantized consensus algorithm that requires nodes to exchange quantized messages and converges in a finite number of steps [40]. This integration helps in significantly reducing the communication load while maintaining the algorithm's overall performance.

The application of quantization in FedAvg is to increase efficiency has been the subject of several studies. For instance, a recent study by [42] shows that FL models that are significantly more robust to different bit-widths during on-device inference are produced by incorporating quantization robustness into the training procedure. Quantization is a strategy that leads to smaller model sizes and lower computational demands by reducing the number of bits needed to describe model parameters. This is especially useful for resource-constrained client devices, like those utilized in FedAvg setups.

In a similar vein, the study by [43] explores various quantization techniques to reduce communication overhead in federated learning, particularly with the FedAvg algorithm. The study highlights the motivation for quantization, its impact on convergence rates, and robustness against quantization noise. Experimental results demonstrate that quantized FedAvg can achieve similar accuracy to unquantized versions with significantly lower communication costs. Additionally, the paper [43] addresses the application of these techniques to heterogeneous devices, enhancing efficiency across diverse participants.

In Federated averaging, gradient quantization takes palce after local training of each clients before gradient are sent back to the server for the aggregation. Similarly quantization emerges as a promising technique within push-pull algorithms. Several studies have explored the application of quantization within push-pull frameworks to optimize performance and reduce computational demands.
At each optimization step, each node executes (i)gradient descent step, subtracting the scaled gradient from its current estimate, and (ii) a finite-time calculation of the quantization average of all nodes' estimates in the network. As a result, this technique closely resembles the centralized gradient descent algorithm. The approach is demonstrated to asymptotically converge to a neighborhood of the best solution at a linear convergence rate [40]. This ensures that the push-pull method remains effective even with the reduced precision due to quantization.
Both FedAvg and push-pull algorithms benefit from quantization, though their im-

plementations and benefits differ. Quantization in FedAvg lowers communication cost during local model aggregation from distributed devices, allowing for effective model updates throughout a mobile device federated network. Because of these devices' limited bandwidth and computational capacity, quantization is essential for maintaining efficiency.

In contrast, the push-pull algorithm employs quantization to manage communication constraints within distributed optimization tasks. Here, quantization focuses on reducing the data exchange between nodes to address the bandwidth limitations inherent in large-scale distributed systems.

To evaluate the effectiveness of quantization in FedAvg and Push-Pull, we conducted experiments using the MNIST and CIFAR-10 datasets. These experiments were designed to measure the impact of various quantization levels(1,2,4,8,16) on the performance and efficiency of the FedAvg and Psh-Pull algorithm. The high-level findings indicate that quantized FedAvg can achieve nearly the same accuracy as the non-quantized version while significantly reducing communication time, though convergence time may increase with higher levels of quantization However, accuracy for Push Pull is decreased and also communication times for lower bits of quantization for both datasets, deatil about the effects of quantization on both algorithms is in the section 3.2

2.4 Erasure

In networking or data transmission, a "missing packet/Erasure" occurs when a data packet, a small unit of data sent through a network, does not reach its intended destination. This might happen due to various factors such as network congestion, transmission issues, and signal deterioration. The absence of packets may lead to communication disruptions, decreased efficiency, or unfinished data transmission. Commonly used methods like packet re-transmission or error correction are employed to reduce the effects of missing packets in protocols that depend on accurate and full packet delivery.

Communication between nodes (devices) and the central node (CN) in distributed systems, particularly in Federated Learning [44], frequently takes place over unstable channels that might cause data loss, known as erasure channels. In an erasure channel, updates or gradient vectors transmitted from a node to the CN are lost or "erased" at a specific probability p_{erase}. When this happens, the CN discards the missing updates and only includes the received, non-erased updates. This method reduces the need for comprehensive information and allows the optimization process to proceed despite communication faults, but with some influence on convergence and speed.

2.4.1 Effects of Erasure

The concept of "erasure" in communication systems involves data loss during transmission, with the receiver being informed of the specific lost data. Unlike errors in bits that lead to receiving inaccurate data, erasures signify the absence of data altogether, impacting communication time, convergence time and accuracy significantly. This part discusses the influence of erasure on system performance and various methods to minimize its effects.
Erasure affects the convergence and efficiency of communication systems in several ways:

2.4.1.1 Impact Convergence time

Missing data may lead to multiple retransmissions, which can result in a slowdown of the communication process as a whole. This issue becomes particularly problematic with protocols that depend on acknowledgements, as the sender must wait for confirmation of successful receipt before transmitting the next packet. Increased erasure rates can lead to significant delays in the acknowledgment process, resulting in delayed convergence of the algorithm which in-turn needs more loops for convergence.

Erasure channels, where updates are lost during transmission at a specific probability p_{erase},directly impact distributed learning algorithms such as Federated Learning (FL). This channel result in incomplete gradient aggregation at the central node, as updates that are lost are not considered in the model updates. consequently, the model's updates are based on partial information, delaying convergence and requiring more iterations to achieve optimal performance. [44] therefore increase in convergence time.

2.4.1.2 Increased Communication Overhead

While erasure typically increases communication overhead due to re-transmissions or redundancy, in some systems, it might paradoxically reduce communication time.This can happen if the system chooses to continue with partial data to maintain speed, or if it sends less data overall as a result of dropped packets. However, a shorter communication time does not always translate into a more efficient system there could be other costs involved, like a slower rate of convergence (requiring more loops for optimal solution) or less precision [44].

2.4.2 Strategies for Mitigation

There are multiple strategies that can be utilized to lessen the impact of erasure in communication systems. Data packets are lost during transmission due to network congestion, device failure, or interference, resulting in missing packets. TCP and other re-transmission protocols are used to mitigate the effects of missing packets by resending all of them. An error-checking code is sent along with the data to correct any problems that may arise during transmission. Duplicate data is sent so that

the recipient can reconstruct the original data if any packets are missing. Quality of Service (QoS) helps by assigning higher priority to certain types of traffic, reducing the likelihood of essential data packet loss. Network redundancy, such as using multiple paths for data delivery, reduces the likelihood of packet loss.

Furthermore, the use of buffering and latency variation can help to smooth out discrepancies in packet arrival timings, lowering the impact of packet loss in live applications. Finally, error detection and recovery methods such as check-sums and CRCs ensure the identification and re-transmission of any damaged or lost packets. Although the major focus is on broader data protection and redundancy solutions, these principles are also useful in managing and eliminating missing packets in network connections. While the strategies mentioned abovesuch as re-transmission protocols, error correction codes, QoS, and network redundancyare vital for minimizing the impact of erasure.
our study specifically focuses on the effects of erasure on communication time, convergence time, accuracy, and convergence loops. Understanding these effects is crucial for developing more efficient algorithms and systems that can better tolerate erasures without relying solely on mitigation techniques.

2.4.3 Erasure in Fedavg and Push-Pull

Missing data can greatly impact distributed machine learning methods like Federated Averaging (FedAvg) and Push-Pull. In federated learning (FL), particularly in wireless networks, the presence of faulty communication channels adds heterogeneity to the system. These communication issues, like as packet loss or transmission faults, can severely disrupt the learning process by sending incomplete or inaccurate model updates to the central server.This disruption is especially significant in algorithms such as FedAvg, which implies that all client updates contribute evenly and consistently to the overall model. However, in environments involving unreliable communication, FedAvg may struggle to maintain stable convergence because missing or corrupted updates can cause considerable variations in the model's training trajectory [45].

The paper [44] explores the challenges and solutions for federated learning (FL) in environments where communication links between clients and the server are unreliable and prone to packet erasure. In such settings, the communication links are modeled as packet erasure channels, where local updates from clients can be lost with a certain probability. This does affect the learning process because some update are lost during communication. This may potentially slow down the convergence of the algorithm and also reduce accuracy . However, it also shows that the impact on convergence can be significantly reduced by using the strategies they propose, such as reusing the last received update when a new update is lost. These strategies help to maintain a convergence rate that is close to what would be achieved in a scenario with perfect communication, thereby demonstrating that federated learning can be robust even in the presence of communication errors. Similarly, Erasures in the decentralized Push-Pull protocol can lead to false updates spreading through

the network, changing the local models and slowing down convergence at each node. Nodes missing crucial updates from neighbors due to lost packets can lead to inconsistencies and potentially hinder the overall optimization process.These effects highlight the critical role that reliable communication plays in maintaining model accuracy and ensuring efficient convergence in federated learning systems.

We consider a scenario where the sender communicates with the receiver over an erasure channel, with each action index sent by the learner potentially being erased with probability ϵ. The receiver is aware of these erasures, but the sender is not, which can lead to suboptimal learning if the receiver does not receive the intended action. This issue mirrors challenges in decentralized systems, such as the Push-Pull protocol, where erasure channels can cause lost messages between nodes. These lost messages can lead to inconsistencies, delayed convergence, and potentially incorrect outcomes in distributed optimization processes. Reliable communication is thus crucial for both effective learning and maintaining accuracy in federated learning[46].

In our work, we simulated the effects of erasure channels on both the FedAvg and Push-Pull algorithms to study their impact on convergence loop and time, accuracy and communication time . By introducing a controlled probability of erasure ϵ in the communication between clients (or nodes) and the central server for fedavg and between clients for Push-Pull algorithms, code implements a stochastic gradient erasure mechanism where each node (except the central node) can have its gradient erased (set to zero) with a certain probability, denoted as p_{erase}.

For FeadAvg, where the central node (CN) aggregates only the available (non-erased) updates from the devices without waiting for full gradients or reusing past updates [44]. A similar approach applies to the Push-Pull algorithm, where nodes aggregate only the received updates without waiting for missing data, maintaining progress despite communication interruptions. We were able to observe how these algorithms perform under real-world conditions where communication is unreliable. The results, which are detailed in the section 3.2 , demonstrate the varying degrees of robustness in these algorithms. For instance, while FedAvg showed sensitivity to missing updates, resulting in delayed convergence time and loop and lower accuracy, Push-Pull exhibited challenges in maintaining consistency across nodes.

The process of gradient aggregation with erasure in a distributed learning setup can be represented as follows:

Gradient Vector and Erasure Probability:
Gradient Vector: Let $\mathbf{g}_i$ be the gradient vector of the i-th node.
Erasure Probability: Let p_{erase} be the probability of erasure.
Erasure Process: For each node i (except the central node), the gradient vector is set to zero with probability p_{erase}:

$$\mathbf{g}_i' = \begin{cases} \mathbf{g}_i & \text{with probability } 1 - p_{\text{erase}} \\ \mathbf{0} & \text{with probability } p_{\text{erase}} \end{cases}$$

Central Node Aggregation: The central node aggregates the gradients from all nodes. If $\mathbf{G}$ is the matrix of gradients from all nodes, the aggregated gradient $\mathbf{G}_{\text{agg}}$ is:

$$\mathbf{G}_{\text{agg}} = \sum_{i=1}^{N} \mathbf{g}_i'$$

where N is the total number of nodes.

our code uses probability-driven approach to simulate communication failures by erasing gradients, star topology for FEDAVG and peer-to-peer for push-pull algorithm by setting the gradient of non-central nodes to zero with a certain probability for FEDAVG In contrast, for the Push-Pull algorithm, the erasure applies uniformly, reflecting equal communication unreliability across all nodes. This approach allows testing the robustness of distributed optimization algorithms against communication failures providing insights into their performance under realistic, unreliable communication conditions.

3

Methods and Results

3.1 Methods

This section investigates the framework that underlies the metrics and important factors to consider our implementation which provides foundation for the subsequent analysis of our experimental results.
In this study, we focused on implementation of federated averaging algorithm and push-pull algorithm with specific attention to the data partitioning strategies, quantization of gradients and erasure of gradients. We also emphasized to analyze the impact of different network topologies on these metrics.

Tools and Libraries: The implementation leverages PyTorch, a popular deep learning library for dataset pre-processing, model definition, training, gradients synchronization and evaluation. Communication between clients during distributed training is managed using MPI (Message Passing Interface) through PyTorchs torch.distributed package. This setup allows for efficient communication and synchronization across multiple clients participating in the federated learning process.

Quantization Type: The implemented code utilizes uniform quantization of gradient values during the synchronization phase. This process involves scaling the gradients using a fixed scaling factor based on the number of quantization bits specified (e.g.,1,2 4, 8, 16 bits). The scaled gradients are then rounded to the nearest integer and clamped within a specified range, ensuring that the quantization process is both efficient and effective in reducing communication overhead. The quantization applied in our framework is specifically during the gradient synchronization phase after the training process. This means that while the model weights are updated in floating-point precision, the gradients communicated between clients are quantized. This form of gradient quantization helps in reducing the bandwidth required for gradient exchange without significantly impacting the model's performance.

Erasure: We have simulated the occurrence of data loss during communication by inducing erasure based on a probability factor. This probability represents the likelihood that the gradients (model updates) from client nodes will be lost or corrupted before they reach the server. During each communication round, a random number is generated and compared against the probability factor. If the random number falls within the probability range, the gradient is erased (not sent), simulating a failed transmission. This approach is done to simulate a real time scenario where

where communication channels are unreliable, such as in networks experiencing high traffic congestion, poor signal strength, or intermittent connectivity.

Network Topology: Different topologies were considered for our implemetation of FedAVg and push-pull (star,Fully-connected,peer to peer, circular) and their impacts on the performance metrics of interest, such as convergence time, communication time, and accuracy.

3.1.1 Implementation

3.1.1.1 Architecture of High Performance Computing Cluster:

HPC system consists of Intel Xeon Gold 6130 and Intel Xeon Gold 6338 cores with few NVIDIA GPU cards. It also has a central storage system, a slurm scheduler and a login node. All these components are inter-connected through Infiniband and ethernet. The cluster is accessed through login node through which we can transfer input files to cluster, prepare batch scripts and send slurm script to scheduler. The scheduler then allocates the resources and starts the script on compute nodes.

Steps to schedule model training in HPC system:
First we login to cluster through the login node and prepare a slurm batch script and a python script for training the model. The slurm script specifies the resource needed, including the number of cores, CPU, GPU, wall time and memory . Once submitted, the job is placed in a queue, ordered by priority. Job starts when requested resources are available and then the automatic environment variables inform MPI to run the job. There is also a temporary directory available within each node which can be used during file I/O operations, however this directory is cleaned immediately when the job ends or fails or crashes. For this thesis we have considered nodes Node 1,4 and Tasks 8,16,24. Nodes in an HPC cluster are individual compute units with their own CPUs, memory, and sometimes GPUs.

In this study, we used Node 1 and Node 4 to explore the scalability of our model training. By using different nodes, we tested how the distributed training algorithms (FedAvg and push-pull) scale when moving from a single node (Node 1) to multiple nodes (Node 4). This helps in understanding the performance impacts of scaling across nodes and how well the communication and computation distribute when using more hardware resources.

In this context, tasks refer to the number of clients or processes involved in distributed training protocols like FedAvg and push-pull. Each task can be viewed as a different client in a federated learning setup or a separate process in a parallel computing environment.
Each task in FedAvg represents a client that trains separately on local data before communicating updates to a central server for aggregate. The number of tasks (8, 16, 24) represents the number of clients involved in the federated learning system.

Push-Pull uses tasks to represent clients or processes that connect directly with one

another to exchange gradients or model updates, rather than a centralized server. Increasing the number of tasks examines the communication protocol's ability to manage larger, more complicated interactions.

Data pre-processing:
In this thesis we have used MNIST and CIFAR-10 dataset. MNIST: A dataset of grayscale images containing 60,000 training samples and 10,000 test samples. Each image represents a handwritten digit (0-9). CIFAR-10: A dataset of RGB images containing 60,000 samples split into 50,000 training samples and 10,000 test samples. It consists of 10 different classes such as airplanes, cars, birds, etc.
Once the scheduler allocates the requested resources (nodes and tasks as specified in the SLURM batch script), the job is sent to the compute nodes. The job script is replicated on other nodes as per the SLURM configuration, allowing independent execution on each computing node. Each Python script is executed independently on its computing node while the communication between nodes is established through MPI (Message Passing Interface). MPI ensures seamless data exchange, synchronization, and coordination between the nodes during training. The Python script initially downloads the dataset (MNIST or CIFAR-10) to a central storage location accessible by all nodes. The dataset is divided according to predefined classes, transforming each data sample into tensor format suitable for neural network training. From the 60,000 samples/images of data, 10,000 images are reserved for testing. These test samples are not exposed to the model during training, ensuring unbiased evaluation. MNIST datsets, images are grayscale, whereas CIFAR-10 datastes images are RGB, requiring different preprocessing steps for image normalization and data augmentation depending on the dataset type.

Data Distribution Across Nodes:
Each computing node receives a unique subset of the dataset using the DistributedSampler, which ensures that the data is randomly and identically distributed across all nodes (IID). This means that each node gets a balanced portion of the data that reflects the overall dataset distribution.
In our implementation, the data across nodes is IID because each node's data subset is randomly sampled and contains a similar distribution of classes and features as the entire dataset. This arrangement differs from real-world federated learning scenarios, in which data is frequently non-IID due to client-specific features. To achieve this IID distribution, we employed a data partitioning strategy that involves shuffling the dataset before dividing it among clients. The use of the PyTorch DistributedSampler with shuffle=True ensures that each client receives a randomly selected subset of the data, maintaining the IID property. The IID sampling approach employed in this implementation ensures that each node processes statistically similar data, resulting in a controlled environment for evaluating distributed training without the additional complication of non-IID data, serving as a baseline for assessing the performance of the federated learning model under ideal data distribution conditions.

CNN Model :
The model considered for training is a 2 convolution layered CNN model with num-

ber of classes as per the dataset used. The dataset and the model is loaded into the computing node then the image samples are passed through each layers of the model which generates the loss. This step is iterated through all the image samples available to each node. An Epoch is defined as an iteration where the model has trained through complete dataset once without repeating. The model is then trained some few epochs, after few epochs we have the upadated model parameters at each computing node. At this point the nodes need to exchange the model parameters to update the model and in order to achieve this MPI calls are used to send the gradients from all the nodes to a single node to avergae the gradients and update the model. Since we have used a star topology for node placements, the central node requests the gradient from all the participating nodes, averages the gradients and updates the model by sharing the new gradients to all nodes including itself. This procedure of training the model for few epochs, gradient averaging, model update and then training again keeps reducing the loss indicating the model is learning effectively and improving its ability to recognise the images. After a certain point the loss does not reduce significantly indicating the model is converged and no additional improvement can be achieved so we stop the training and log the evaluation metrics for further evaluation.

Testing the model:
Once the model is trained, it is tested for its performance using the image samples which were not used during the training step. By doing this we evaluate the performance of model with a separate set of dataset. The purpose here is to assess how the model works with unseen images providing a correlation to its real world performance. If the model performance good on training images but does not perform well on testing images it means model is Overfitting where model has learned the images but not the underlying generalised information. Similarly if the model performs poorly on both training and test images , it indicates underfitting signifying model is simple for dataset used.

3.2 Results

This section provides a detailed analysis of the experimental results obtained from applying different quantization levels, erasure and network topologies to the FedAvg and Push-Pull algorithm on the MNIST and CIFAR-10 datasets.
The FedAvg algorithm was implemented using a star topology, where a central server aggregates the model updates from all clients, making it the default approach in centralized federated learning setups.
On the other hand, the Push-Pull algorithm, typically used in distributed optimization, was applied in a peer to peer connected topology, where each client directly communicates with others in a peer-to-peer network. In this setup, the Push-Pull algorithm facilitates a decentralized learning process, allowing clients to collaboratively train a model without a central server. Both algorithms were analyzed under various quantization levels to assess their performance on the MNIST and CIFAR-10 datasets.
The quantization method used here is uniform quantization, which scales the gradi-

ent values to a fixed range based on the number of bits and then rounds them to the nearest integer. The quantization for 1-bit gradients is a special case, using the sign function.
The metrics considered for evaluation are defined in section 3.1 .

3.2.1 Analysis of Quantization Impact on FedAvg

3.2.1.1 Accuracy

For the MNIST dataset, the accuracy remains high across all quantization levels, with a slight decrease observed at 1-bit quantization. Specifically, the accuracy decreases from 98.42% with no quantization to 97.25% with 1-bit quantization due to reduction in precision which introduces noise. The accuracy stabilizes around 98% for higher bit quantization levels (2-bit, 4-bit, 8-bit, and 16-bit), indicating the robustness of the model to quantization.
For the CIFAR-10 dataset, The accuracy of the model slightly decreases as the quantization level decreases. The accuracy drops from 80% with no quantization to 77.56% with 1-bit quantization, indicating a minor loss of precision. However, the accuracy stabilizes around 79-80% for higher bit quantization levels. This behavior is expected as lower bit quantization introduces more approximation errors, but higher bit levels maintain sufficient precision to preserve model performance.
While both datasets exhibit a drop in accuracy at the lowest bit level, the difference in the behavior of accuracy drop is due to the complexity of the datasets, the nature of the images, and the model's sensitivity to quantization noise. However, the accuracy stabilizes at higher bit levels, demonstrating that quantization can be applied effectively with minimal loss of performance, especially at bit levels that strike a balance between precision and efficiency.

3.2.1.2 Average communication Time

The average communication time for MNIST datasets decreases significantly with lower bit quantization levels. The communication time reduces from 943.60 seconds with no quantization to 553.38 seconds with 1-bit quantization, representing a reduction of approximately 41.36%. As we increase the bit width, the communication time gradually increases, achieving 714.19 seconds with 2-bit quantization (24.33%), 718.66 seconds with 4-bit quantization (23.82% reduction), 816.75 seconds with 8-bit quantization (13.44% reduction), and 806.25 seconds with 16-bit quantization (14.56% reduction).
The average communication time for CIFAR-10 dataset generally decreases with lower bit quantization levels. The communication time reduces from 8591.18 seconds with no quantization to 7541.56 seconds with 1-bit quantization, representing a reduction of approximately 12.22%. However, it's noteworthy that the 1-bit quantization does not achieve the lowest communication time among all levels. As we increase the bit width, the communication time further reduces, achieving 7236.27 seconds with 2-bit quantization (15.79% reduction), 7513.10 seconds with 4-bit quantization (12.55% reduction), 7518.80 seconds with 8-bit quantization (12.50% reduction), and 7737.53 seconds with 16-bit quantization (9.92% reduction).

The slightly higher communication time observed with 1-bit quantization CIFAR-10, compared to 2-bit, may be attributed to the computational overhead associated with quantizing and dequantizing the model parameters at such a low bit-width. The process of handling these very small data packets might introduce inefficiencies that offset some of the gains from reduced data volume. Additionally, network and implementation specifics, such as error handling and retransmissions due to quantization noise, could also play a role.These insights are supported by findings in related works such as [47] who highlighted the trade-offs involved in gradient compression methods like signSGD, where reducing data volume can sometimes introduce inefficiencies that counterbalance the gains and [48] which discuss the challenges and overheads associated with low-bit quantization.Communication time reduction was more significant for MNIST due to its simpler nature, while CIFAR-10's complexity introduced overhead at 1-bit quantization.
In large-scale federated learning deployments, even small percentage gains can lead to significant absolute time savings. The results demonstrate that using lower bit quantization levels can substantially improve communication efficiency without severely compromising the models performance. These findings underscore the practical benefits of quantization in enhancing communication efficiency in federated learning, validating our approach and encouraging further research into optimal quantization levels for various distributed learning scenarios.

3.2.1.3 Execution Time

Execution time is indeed a crucial metric for assessing the efficiency of machine learning models.The data clearly shows that execution times are typically reduced by lower quantization levels, most likely as a result of the decreased computational needs of handling smaller numerical representations.
The execution time shows a decreasing trend with lower bit quantization for both the MNIST and CIFAR-10 datasets. For MNIST, it reduces from 103.67 minutes without quantization to 98.57 minutes with 1-bit quantization. For CIFAR-10, the time drops from 585.22 minutes without quantization to 518.04 minutes with 2-bit quantization. However, the 1-bit quantization for CIFAR-10 results in a relatively higher execution time due to the need for more communication rounds to compensate for reduced gradient precision in this complex dataset. Despite some variation at higher bit levels, the overall reduction in execution time demonstrates the computational efficiency gained through quantization.

3.2.1.4 Average Convergence Time and Convergence Loop

Convergence time tends to decrease slightly with lower quantization levels, but the differences are less pronounced compared to communication time. Both datasets show similar convergence time patterns, with MNIST converging faster due to its simplicity, while CIFAR-10 requires more time due to its complexity. The convergence loop, represented by the number of FedAvg loops, indicates the number of communication rounds required for the model to converge to a stable state.
Quantization, reduces the precision of the model updates sent from clients to the server, can introduce noise and affect the convergence rate. Lower bit quantization

(e.g.1-bit) can significantly slow down convergence due to the high noise in gradient updates. However, studies [49] have shown that with careful design, such as using higher bit quantization or adjusting the learning rate, the convergence rate can be improved and made comparable to full precision .
The difference in convergence behavior between datasets such as CIFAR-10 and MNIST is primarily due to the complexity of the datasets. CIFAR-10, with its colored images and intricate features, tends to have a slower convergence rate compared to the simpler grayscale digit images in MNIST. The sensitivity to quantization noise is higher in CIFAR-10 due to its complex feature space, leading to more pronounced impacts on convergence.
The convergence loop for CIFAR-10 dataset remains relatively stable across different quantization levels, with minor variations. The convergence rate is 20 FedAvg loops for no quantization and increases slightly to 21 loops for 1-bit and 2-bit quantization, then stabilizes at 19 loops for higher bit levels. This stability indicates that the FedAvg algorithm can handle quantization noise without significantly affecting the convergence loop. In contrast, for MNIST dataset, being less complex, shows greater resilience to quantization noise. The convergence rate for MNIST remains stable, requiring 9 FedAvg loops without quantization, slightly increasing to 11 loops with 1-bit quantization, and stabilizing at 10 loops for higher bit levels.
The convergence loop analysis highlights the trade-off between quantization level and algorithm performance. While lower-bit quantization can lead to minor increases in convergence loops, the benefits in communication efficiency and execution time often outweigh these costs. These findings emphasize the importance of selecting appropriate quantization levels to balance communication overhead with convergence performance, particularly in federated learning scenarios where communication costs are a significant consideration. Future research should focus on adaptive quantization strategies that dynamically adjust bit-widths to optimize convergence rates based on dataset complexity and network conditions. The quantization bits and their effects on metrics shown in figure 3.1. Quantization effectively reduced communication time and execution time without severely impacting accuracy or convergence. While lower bit quantization generally improved efficiency, the optimal bit level might vary based on specific dataset and model characteristics.

3.2.2 Analysis of Quantization Impact on Pushpull

3.2.2.1 Accuracy

The MNIST and CIFAR-10 datasets were examined through a push-pull method (peer-to-peer) with varying levels of quantization and configurations, including 8 clients and no quantization. Accuracy for the MNIST dataset rose from 76.35% using 1-bit quantization to 97.62% with 8-bit quantization, then decreased to 97.14% with 16-bit quantization. The configuration of 8 clients and absence of quantization resulted in the top accuracy of 98%. Accuracy for the CIFAR-10 dataset increased from 79% using 1-bit to 82% with 4-bit quantization and stayed consistent up to 16-bit quantization, with no quantization achieving above 82%.

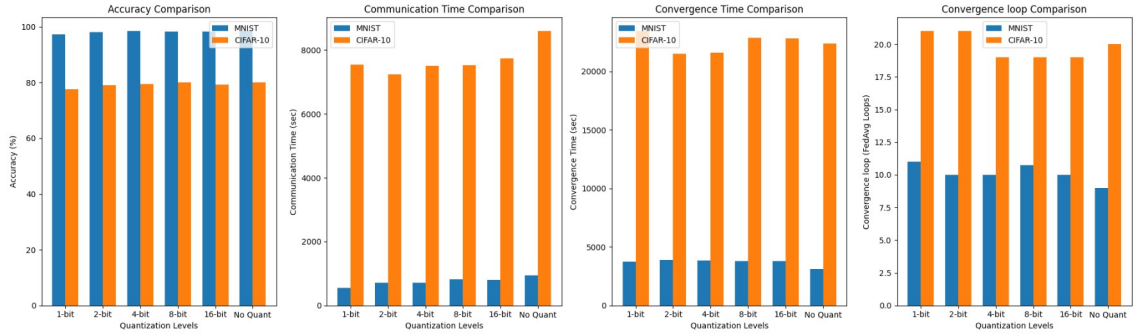

Figure 3.1: Impact of Quantization Levels on Accuracy, Communication Time, Convergence Time, and Convergence Rate for MNIST and CIFAR-10 Datasets. The leftmost subplot compares the accuracy percentages of the datasets across different quantization levels. The second subplot shows the communication time in seconds, highlighting the efficiency of data transfer. The third subplot presents the convergence time in seconds, reflecting the overall time required for the models to converge during training. The rightmost subplot shows the convergence rate, measured in FedAvg loops, indicating the number of communication rounds required for the model to converge to a stable state.

3.2.2.2 Average communication Time

In this paper [40],consider the unconstrained distributed optimization problem, in which the exchange of information in the network is captured by a directed graph topology, thus, nodes can only communicate with their neighbors. Additionally, in our problem, the communication channels among the nodes have limited bandwidth. In order to alleviate this limitation, quantized messages should be exchanged among the nodes. For solving this distributed optimization problem, we combine a gradient descent method with a distributed quantized consensus algorithm (which requires the nodes to exchange quantized messages and converges in a finite number of steps). Specifically, at every optimization step, each node (i) performs a gradient descent step (i.e., subtracts the scaled gradient from its current estimate), and (ii) performs a finite-time calculation of the quantized average of every nodes estimate in the network.

In Koloskova et al. (2019), the authors present a gossip-based stochastic gradient descent algorithm, which utilizes arbitrary compressed messages and exhibits linear convergence.The main idea of quantization in this paper is that nodes transmit a compressed value (i.e., quantized) of their stored information as they require a few bits for representation compared to the non-compressed ones (i.e., real values) which in theory require an infinite number of bits. For this reason, communication-efficient distributed optimization has received significant attention recently in the control and machine learning communities.

As a result, the impact of quantization on push pull algorithm for MNIST dataset is that the communication times for no quantization is 917.97 seconds. Nevertheless, the transmission latency fluctuates significantly depending on the number of quantization bits employed. By using 1-bit quantization, the communication time is reduced significantly to 141.67 seconds. The notable decrease can be attributed to

the acceleration of communication time with 1-bit quantization, which reduces the amount of data exchanged among peers. Nonetheless, the time for communication sharply increases to 1118.22 seconds when the quantization bit count is raised to 2-bit. The sudden increase suggests that the benefits of smaller data sets are less important than the challenge of managing slightly more accurate quantization. The durations for communication when quantizing with 4 bits, 8 bits, and 16 bits are as follows: 584.68 seconds, 384.22 seconds, and 688.08 seconds, respectively.
The varying levels of quantization present a different perspective on the communication times for the CIFAR-10 dataset. The duration of communication is 5380.20 seconds, much longer when not quantized. While not as dramatic, quantization boosts communication times in the MNIST dataset, as seen. The communication time decreases to 5132.59 seconds when using 1-bit quantization and for varying quantization bit: 2, 4, 8, and 16. The communication times vary: 5310.19, 5332.63, 5432.17 and 5459.84 seconds.
As the number of quantization bits increases (2, 4, 8, 16 bits), communication time gradually increases from 5310.19 seconds to 5459.84 seconds for CIFAR10 and MNIST from 141.67 to 688.08 seconds . Although higher-bit quantization provides more accurate data precision, it increases the data size and introduces additional computational overhead, leading to longer communication times.

3.2.2.3 Average convergence time and convergence loop

The impact of different quantization levels on the convergence times of machine learning models trained with a push-pull strategy on the MNIST and CIFAR10 datasets. The original convergence time of the MNIST dataset without quantization is 1147.62 seconds. By implementing 1-bit quantization, the time is significantly shortened to 944.89 seconds, indicating a substantial enhancement as a result of the extensive data compression, enabling faster communication. Yet, increasing the quantization to 2 bits results in a substantial increase in convergence time to 2118.91 seconds, showing a less effective trade-off between communication efficiency and accuracy of quantized values. Convergence times for quantization with 4 bits, 8 bits, and 16 bits are 1498.69 seconds, 1507.58 seconds, and 1423.52 seconds, showing varying levels of efficiency compared to each other and the baseline.
The CIFAR10 dataset has a baseline convergence time of 19249.72 seconds when no quantization is applied. With 1-bit quantization, there is an improvement seen as the convergence time decreases to 18725.38 seconds.With convergence lengths of 18935.39 and 18925.96 seconds, respectively, the 2-bit and 4-bit quantisation levels are marginally longer than the baseline but shorter than the 1-bit quantisation time. The convergence times for the 8-bit and 16-bit quantisation levels are 19165.05 seconds and 19146.30 seconds, respectively.

Our study investigated the impact of quantization in distributed machine learning contexts, specifically how different quantization levels and network structures affect the number of convergence loops needed when dealing with datasets such as MNIST and CIFAR-10. For instance, the comparison of a push-pull (peer-to-peer) quantization approach in a distributed machine learning scenario revealed that varying the number of quantization bits, which dictate data transmission accuracy, does

not significantly affect the convergence loops. Specifically, for the MNIST dataset, quantization levels ranging from 1 to 16 bits resulted in a consistent convergence rate of 8 loops, only slightly lower than the 9 loops observed for a non-quantized configuration. Similarly, the CIFAR-10 dataset exhibited a convergence rate of 17 loops with 16-bit quantization, just below the 18 loops seen in full-precision setups. These findings suggest that decreasing data precision through quantization can effectively reduce transmission costs without significantly impacting convergence speed, aligning with the efficiency goals seen in the earlier works by Pu et al. and Xin and Khan[40][50] .Ansd also with Basu et al. (2019), the authors present a distributed optimization algorithm which combines aggressive sparsification with quantization. The algorithm keeps track of the difference between the true and compressed gradients, and converges with equal convergence rate as its non-quantized version.

3.2.2.4 Execution Time

The recorded time it takes to complete different levels of quantization demonstrates the balance between smaller data size and the additional computational work involved in the quantization process. When data is not quantized, it remains at its full precision, causing slower execution times because of the high communication overhead needed to transmit and process all the information. Utilizing 1-bit quantization leads to a dramatic reduction in data size, leading to quicker communication and shorter execution times, as seen in both the MNIST and CIFAR10 datasets. Yet, as the quantization bits go up to 2, 4, 8, and 16 bits, the execution times start to rise. This 3.2 displays how execution time benefits from reduced data size with low-bit quantization, but as quantization levels increase, performance gains decrease due to higher computational and communication overhead.

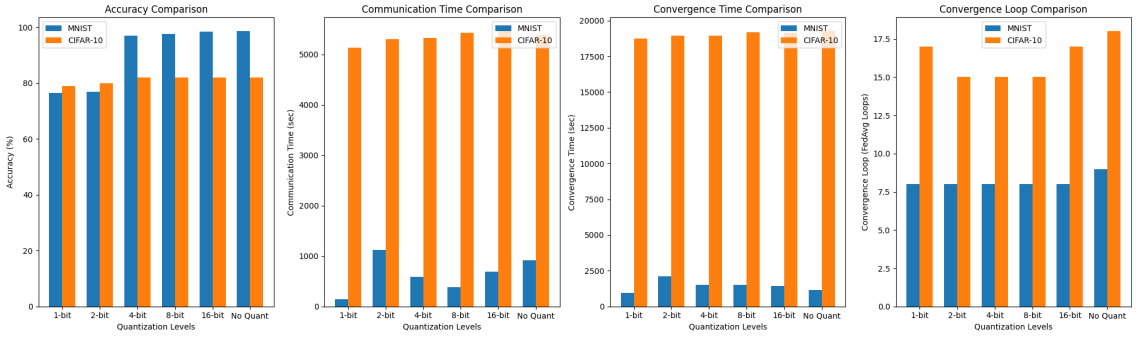

Figure 3.2: Impact of Quantization Levels on Accuracy, Communication Time, and Convergence Time for MNIST and CIFAR-10 Datasets. The left subplot compares the accuracy percentages of the datasets across different quantization levels, indicating the effect on model performance. The middle subplot shows the communication time in seconds, highlighting the efficiency of data transfer. The right subplot presents the convergence time in seconds, reflecting the overall time required for the models to converge during training.

3.2.3 Analysis of Erasure Impact on FedAvg

The results obtained from the experiments with both the MNIST and CIFAR-10 datasets, conducted under conditions of packet erasure, reveal several important insights.
For the MNIST dataset, the slight decrease in accuracy from 98.42% (baseline) to 98.07% under a 0.4 erasure rate. Similarly for CIFAR-10, accuracy decreased slightly from 80% (baseline) to 79.63% with 0.4 erasure rate, indicates that while erasures do impact the learning process, the system remains relatively robust even with some data loss. This minimal accuracy drop suggests that the overall learning process can tolerate some degree of erasure without significant degradation, as highlighted by [44]. The paper also discusses how federated learning algorithms can maintain convergence and accuracy despite communication errors, which aligns with the stability observed in our results.
Furthermore, the decrease in communication time observed in both datasets-from 943.605 seconds to 558.79 seconds for MNIST and from 8,591.18 seconds to 6,183.78 seconds for CIFAR-10, despite the presence of erasures, also aligns with the paper's findings that communication overhead can be reduced in such scenarios. However, as [44] notes, this reduction in communication time does not necessarily lead to a more efficient system overall. As seen in our results, for instance, convergence time increased from 3,109.45 seconds to 4,063.81 seconds for MNIST and from 22,383.83 seconds to 24,622.21 seconds for CIFAR-10, while convergence loops increased from 9 to approximately 12 for MNIST and from 20 to 22 for CIFAR-10. Additionally, we observed a slight decrease in accuracy across both datasets. This increase in convergence time and loops is due to erasures as it leads to missing client updates, which requires more rounds to compensate for missing updates and achieve stable accuracy

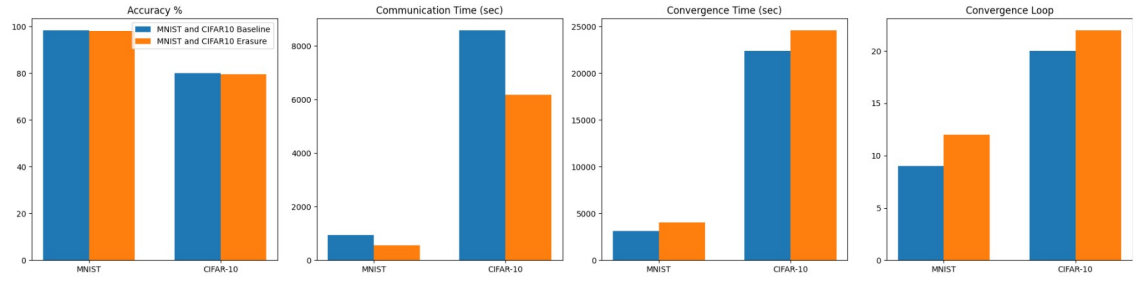

Figure 3.3: Impact of Erasure on Accuracy, Communication Time, Convergence Time and loop for MNIST and CIFAR10.

3.2.4 Analysis of Erasure Impact on Pushpull

The goal of the erasure technique is to address potential data loss commonly observed in distributed systems with unreliable node connectivity, enhancing the resilience and fault tolerance of the system. Nevertheless, this strength is accompanied by specific processing costs and a slight decrease in accuracy. When using the MNIST dataset, the erasure technique slightly decreases accuracy (from 98% to

97.09%) but also significantly increases convergence time (from 1147.62 seconds to 2355.42 seconds). The increased time for convergence is probably because of the extra processing needed to compensate for possible packet loss, guaranteeing the model's resilience in case of data loss during transmission. In spite of the extended convergence time, this approach manages to enhance communication time slightly (from 917.97 seconds to 847.72 seconds) and reduce total execution time (from 4093 seconds to 3256 seconds). This demonstrates the model's ability to successfully address and minimize the impact of missing packets without causing notable delays in the training procedure. The increase in convergence rate (from 9 to 10) means that the model will need additional iterations to stabilize because the erasure process adds complexity in handling data loss.

Likewise, on the CIFAR-10 dataset, erasure technique led to a notable decrease in accuracy (from 82% to 80%), as a result of the data's higher complexity and dimensionality. The time taken for convergence significantly rises (from 19249.72 seconds to 193045.10 seconds), showing the increased computational burden of managing packet loss in a more intricate dataset. However, the communication time slightly enhances (from 5380.20 seconds to 5158.98 seconds) while the execution time decreases (from 25433 seconds to 23279 seconds), showing that the model still profits from training efficiency enhancements. The convergence loop saw a slight rise, going from 18 to 19, emphasizing the extra work needed for achieving stable model performance with the erasure technique.
The study highlights that while communication losses can reduce immediate communication time, they impose a longer-term penalty on convergence loop and time and accuracy, particularly when the erasure is applied.

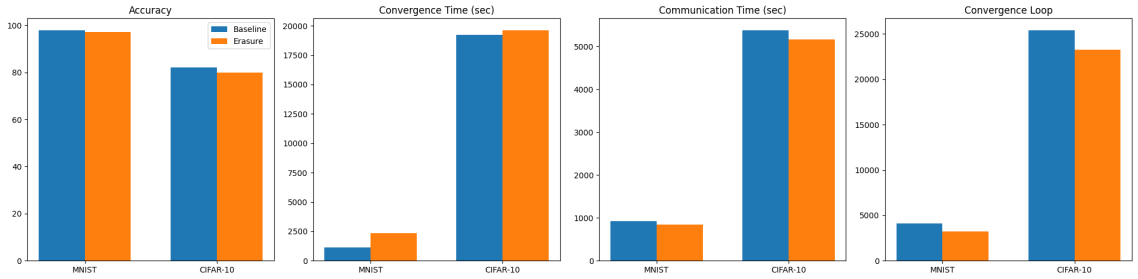

Figure 3.4: The figure illustrates the performance differences between the baseline and erasure methods in a peer-to-peer quantization strategy for both MNIST and CIFAR-10 datasets, particularly in managing missing packets or data loss

3.2.5 Communication Topologies

Understanding communication topologies is crucial in the design of distributed systems, particularly in the implementation of Federated Averaging (FedAvg) and Push-Pull mechanisms. This section explores various communication topologies and how FedAvg and Push-Pull can be integrated effectively.

3.2.5.1 Topology in FedAVg

FedAvg is centralized setup(star topology), Involves a central server that aggregates model updates from all clients. Clients communicate directly with the central server. Additionally, this centralized approach has a significant vulnerability: the central server acts as a single point of failure, making the system more susceptible to communication disruptions.
whereas in decentralized FeadAvg (FC), clients communicate directly with each other, forming a peer-to-peer network without a central server. Communication overhead is more in centralized FedAvg (star) due to frequent and direct communication between clients and server but communication overhead in decentarlized setup is less compared to centralized due to communication is distributed among client, reducing the overall load on any single node but possibly increasing the number of communications needed to reach consensus. The paper [51] primarily focuses on decentralized setups(FC), it indirectly highlights how decentralized learning is more resilient to communication issues compared to centralized approaches, where a single point of failure (the central server) can be more vulnerable to communication disruptions.
A intermediate approach between fully decentralized setups and centralized ones is represented by a circular (or ring) architecture. Each client in this topology forms a ring-like structure by communicating in a closed loop with its immediate neighbors. By reducing the amount of direct connections that each client has, this technique lowers communication overhead and more equally distributes the load throughout the network. By eliminating the central server, it provides greater resilience than centralized FedAvg; nevertheless, due to the possibility of information flow disruption from the failure of a single client or communication link, it is not as resilient as fully connected decentralized configurations.
One example of the use of circular topology is the cyclic federated learning approach[52], which has been investigated in studies involving medical data. This approach is especially suitable for situations where data security and privacy are critical since it places a strong emphasis on knowledge distillation and distribution of information. The circular topology's regulated communication flow makes it easier to handle sensitive data while preserving the system's resilience and effectiveness.

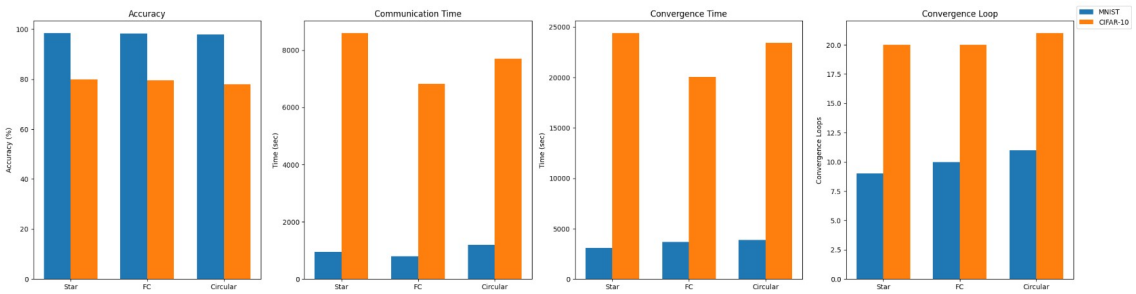

Figure 3.5: Figure presents a comparative analysis of three different algorithm topologies (Star, Fully Connected (FC), and Circular) across two datasets, CIFAR-10 and MNIST

The Figure 3.5 shows that, The accuracy remains consistent across different topologies (Star, Fully Connected (FC), and Circular)for both datasets However,the accuracy of decentralized FedAvg is slightly lower compared to the centralized approach. Specifically, for CIFAR-10, decentralized FedAvg achieves 79.6% accuracy which is 0.5% lower compared to 80% in the centralized method, while for MNIST, the accuracy is 98.34% in the decentralized approach exhibiting a negligible 0.08% decrease compared to 98.42% in the centralized one. This slight decrease in accuracy in the decentralized method may be attributed to the lack of global aggregation, leading to potential inconsistencies in updates across different parts of the network[51].
This suggests that topology selection has minimal impact on model accuracy. However, because FC allows direct client-to-client connection, it eliminates the central server bottleneck and has a shorter communication time than the star topology. Because of the central server bottleneck, the star topology experiences longer communication and execution times, whereas the circular topologywhich distributes updates in a ring-like fashion has intermediate communication and execution duration. Notably, the Circular topology required more convergence loops, suggesting a less efficient and sequential and dispersed communication pattern compared to FC and Star.

3.2.5.2 Topology in Push-Pull

In a star-shaped network, agents communicate through a central hub, influencing communication efficiency and information flow.The central hub is pivotal in distributing information and coordinating updates among the agents.
In a push-pull distributed machine learning framework, the central node plays a crucial role in aggregating information from various base classifiers distributed across different nodes. To enhance this aggregation process, In the central node, we train a probabilistic model to aggregate the base classifiers. We investigate a model relying on conditional probabilities of classifier outputs given the true class of an input (whose estimation can be decentralized without difficulty). These distributions are used as building blocks to classify unseen examples as those maximizing class probabilities given all classifiers outputs [53][54]]. The originality of our approach consists in resorting to copula functions to obtain a relatively simple model of joint conditional distributions of the local base classifier outputs given the true class.
In a master-worker architecture employing a star topology, one node acts as the master (sometimes also called the fusion center), maintaining the authoritative copy of the optimization variable. At each iteration, the master sends to every agent, and each agent iii computes and returns to the master. The master then averages the gradients it receives from all agents, and once it has received gradients from every agent, it performs the gradient-descent update before proceeding to the next iteration[2]. This architecture is relatively simple to implement and aligns with the push-pull paradigm where the master node pushes the current state to all agents and pulls the computed gradients from them.
The Star topology demonstrated the most efficient performance among the algorithms. It achieved an accuracy of 98%, the fastest convergence time of 1338.52 seconds, the shortest communication time of 400.37 seconds, and the lowest execution time of 1760 seconds. This indicates that the Star topology not only ensures

high accuracy but also optimizes both communication and processing times, making it the most effective choice in terms of overall efficiency.

A circular network topology creates a continuous loop for communication among agents, impacting how information spreads and synchronization occurs. Consensus algorithms adjust to this circular layout by utilizing interactions with neighboring agents along the loop. Circular network topology is widely used in various network designs, serving as an extension of the ring topology for implementing local area networks. These networks, also known as distributed loop computer-networks, are referenced in [55] [56]. A variant of the circular network topology, called Multi-Ring topology, is employed to achieve high-performance group communication in peer-to-peer networks. where nodes are connected to their immediate neighbors to form a ring-like structure, the push-pull communication method enhances data transfer efficiency. In the push phase, a node initiates the process by sending data to its adjacent neighbors. These neighbors then forward the data to their own neighbors, creating a ripple effect around the ring. This ensures that the data propagates uniformly and efficiently across the network. Conversely, in the pull phase, a node requests data from its neighbors, who either provide the data if they have it or propagate the request further around the ring until the data is found and returned to the requesting node. This method leverages the uniform distance between nodes in a circular topology, ensuring balanced and reliable communication. By distributing the communication load evenly and preventing any single node from becoming a bottleneck, the push-pull method maintains high performance across the network. For example, in a distributed file system utilizing this topology, nodes can efficiently disseminate updates or retrieve files through coordinated push and pull operations, ensuring all nodes are promptly informed or serviced.
The Circular topology, despite reaching a 98% accuracy like the other topologies, displayed the lowest efficiency in terms of time metrics. Its convergence time was the slowest at 1428.02 seconds while its communication time was the highest at 656.76 seconds. As a result, it took the longest time to execute, totaling 2106 seconds. These findings indicate that while the Circular topology is accurate, it is the least effective in terms of convergence, communication, and overall execution times.

In a fully connected network topology, every pair of agents can communicate directly with each other. Although this setup offers high communication efficiency, algorithms need to address the challenges of managing elevated communication overhead and potential redundancy in information exchange. We assume a fully connected topology allowing each node to share its trained base classifier with every other node as well as with a central node which will aggregate models. Local training phases do not have to be synchronized. Ensemble methods or multiple classifier systems are good candidates to operate in such a form of decentralized learning. Indeed, many such methods do not require that the base learners, i.e. those trained on each local node, have to collaborate at training time [57]. The push-pull framework can be effectively utilized to manage this decentralized learning process. In this framework, each node independently trains its base classifier using its local data. Once a node completes its training phase, it pushes its trained base classifier to all other nodes and the central aggregator. Simultaneously, each node pulls the latest

base classifiers from other nodes, ensuring it has access to the most recent models for ensemble learning.
The Fully connecte(FC) topology achieved 98% accuracy, however, it showed lower efficiency compared to the other topologies. The convergence time was 1358.38 seconds and had a relatively high communication time of 525.48 seconds, leading to an overall execution time of 1915 seconds. Although its performance exceeds the Circular topology, it is not as efficient as the Star topology because of its longer communication time.
Randomly connected networks introduce diversity in communication paths and neighbor connections. Algorithms need to adapt dynamically to these shifting network structures to maintain resilience and flexibility in information exchange. To reduce communication latency, the network topology of a parallel system should have low diameter and low average shortest path length (ASPL)[58]. Random network topologies have been proved to be useful for achieving low diameter and low ASPL, making them advantageous for parallel systems. The communication latency is a significant performance bottleneck for large parallel systems, which often have hundreds of thousands of compute nodes. The latency in communicating across such a large number of nodes can severely impact overall system performance. In the push-pull communication model, where nodes alternately push data to and pull data from their neighbors, the benefits of random topologies are particularly evident. The inherent randomness ensures that the network maintains a low average shortest path length and minimal contention, which is crucial for efficient data dissemination and retrieval. This leads to a significant reduction in communication latency, addressing a critical bottleneck in large parallel systems. The Random topology had equal 98% accuracy but varied performance in time metrics. The convergence time totaled 1391.52 seconds while the communication time was 496.94 seconds. 1922 seconds were required for the Random topology to be executed. While it is more effective than the Circular topology, the Mesh topology lags behind the Star topology in efficiency due to longer convergence and communication duration.
Even though the CIFAR-10 dataset was more complex, the topologies still managed to reach an accuracy of 76%, however their efficiencies varied. The Fully Connected Topology was found to be the most effective, with a convergence time of 3765.42 seconds, communication time of 1089.59 seconds, and execution time of 25,044 seconds. The Random Topology had a successful performance, achieving the shortest communication time of 1075.47 seconds, but with a slightly longer convergence time of 3839.52 seconds and an execution time of 25,361 seconds. The Circular Topology showed a slight decrease in efficiency, with a convergence time of 3779.17 seconds, a communication time of 1095.25 seconds, and an execution time of 25,146 seconds. The Star Topology, which performed well in the MNIST dataset, faced challenges with CIFAR-10, demonstrating a convergence time of 3848.04 seconds, communication time of 1169.61 seconds, and execution time of 25,904 seconds. This difference highlights how the Star Topology, although optimal for basic tasks, is not as suitable for intricate datasets, where the Fully Connected Topology excels. It was discovered that the network topology had a greater impact on convergence speeds. Within the MNIST dataset, different structured topologies like star, fully connected, and circular achieved a convergence rate of 8, surpassing the random topology's rate of 7. In

the CIFAR-10 dataset, the fully connected topology achieves the fastest convergence rate at 19, while the circular and star topologies follow closely behind at 18, and the random topology at 17. These results emphasize the significance of selecting the right network structure, which can greatly enhance training efficiency in distributed machine learning systems, even when communication overhead is controlled through quantization.

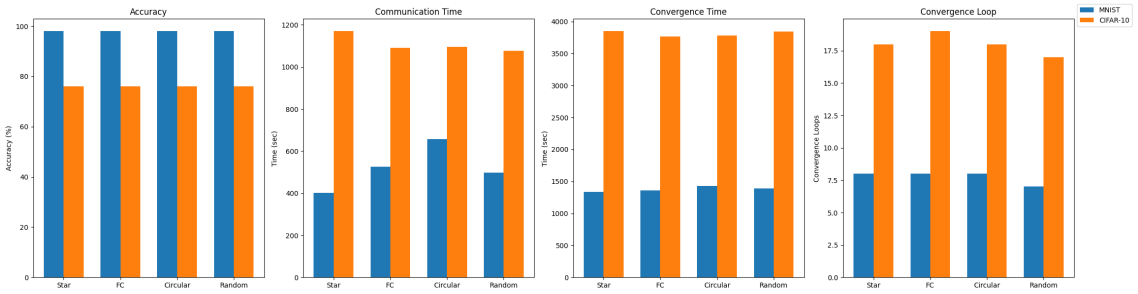

Figure 3.6: Comparison of Different Graph Topologies on Accuracy, Communication Time, and Convergence Time for MNIST and CIFAR-10 Datasets. The left subplot compares the accuracy percentages of the datasets across different graph topologies (star, fully connected, circular, and random), indicating the effect on model performance. The middle subplot shows the communication time in seconds, highlighting the efficiency of data transfer. The right subplot presents the convergence time in seconds, reflecting the overall time required for the models to converge during training

3.2.6 Analysis of Scalability in FEDAVG

Scalability is a important aspect of any distributed learning system, especially in federated learning, where the number of participating clients might range from a few to millions. As the system grows, it must effectively manage rising computing and communication needs while preserving model performance. Scalability in federated learning refers to the ability to manage a rising number of clients without significantly raising communication cost or degrading the model's accuracy and convergence speed.
As we scaled the number of clients, we observed a slight decrease in model accuracy, especially in complex datasets like CIFAR-10. This issue discussed in [32] where they suggest using resilient aggregation strategies like Byzantine-resilient secure aggregation and adaptive algorithms for maintaining accuracy in federated learning as system scales. Implementing such strategies may assist in preventing the accuracy loss as the system scaled.

Our results indicate a significant increase in communication time as the number of clients scales from 8 to 24, which aligns with the observations made by[32] regarding the communication bottlenecks in federated learning. Specifically, in our scalability experiments, communication time increased substantially, highlighting the challenges of scaling federated learning systems. For instance, communication time for the MNIST dataset increased from 696.81 seconds with 8 clients to 1,872.89 seconds with 24 clients. Similarly, on the CIFAR-10 dataset, the communication time increased from 8,591.18 seconds to 9,284.62 seconds under the same conditions. To address the communication bottleneck in a different context, we experimented with gradient quantization techniques, however this was applied only in the scenario of Node 1, Task 8. In this isolated test, quantization led to a decrease in communication time, demonstrating its potential to alleviate some of the communication burdens. However, since this quantization experiment was not conducted across all scalability scenarios, further investigation is needed to understand its broader impact when scaling the number of clients. These separate experimental paths underscore the importance of integrating communication-efficient techniques, such as quantization into broader scalability strategies to fully assess their effectiveness in large-scale federated learning deployments.
Convergence behavior of federated learning system as the number of clients increased from 8 to 24, the convergence time tended to decrease, with the average convergence time for the MNIST dataset dropping from 3,828.43 seconds with 8 clients to 1,957.81 seconds with 24 clients. This trend was accompanied by an increase in the number of convergence loops required to achieve stability, with the convergence loops increasing from 9 to 13.74 as the number of clients grew. For the CIFAR-10 dataset, while a similar pattern of decreased convergence time was observed, with decrease in time from 24,383.83 seconds with 8 clients to 11,891.45 seconds with 24 clients, the number of convergence loops also increased, rising from 20 to approximately 20 as the number of clients scaled. This suggests that while the system converges faster overall, it requires more loops to reach a stable global model as more clients contribute to the training process. Interestingly, this increase of convergence loops did not necessarily correspond with higher model accuracy, especially in the more

complicated CIFAR-10 dataset, where accuracy declined as the number of clients grew, despite the faster convergence time. These findings align with the observations made by [32], discussed that while increasing the number of clients can sometimes accelerate convergence, it can also introduce challenges related to model accuracy and data heterogeneity. Our experiments underscore the importance of carefully balancing the number of local updates, the frequency of communication, and the convergence loops to ensure that convergence is achieved without sacrificing model performance.

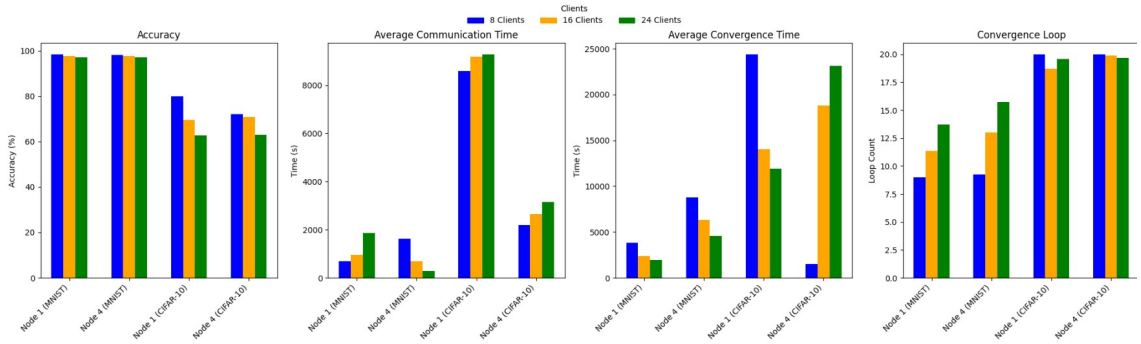

Figure 3.7: Scalability analysis of MNIST and CIFAR-10 datasets showing the impact of increasing clients on accuracy, convergence time, communication time, and convergence loops.

3.2.7 Analysis of Scalability on Push-pull

The scalability properties of hybrid protocols like PoP (Push-or-Pull) and PaP (Push-and-Pull) are notably superior to those of singular approaches like Pull or Push. As highlighted in the study, the protocol PoPoPaP, which intelligently selects among Push, Pull, or PaP based on context, exemplifies this adaptability. This dynamic selection ensures optimal performance across all critical dimensions: resiliency, temporal coherency, and scalability[59]. Building on this understanding, my results from distributed computing experiments further illustrate these principles. Commencing with the MNIST dataseta benchmark comprising 70,000 28x28 grayscale images of handwritten digitsthe scaling outcomes underscore the efficacy of distributed computing. In configurations such as N1T8, N1T16, and N1T24 (where 'N1' denotes a single node and 'T' the number of tasks per node), the model consistently achieves 98% accuracy. This uniformity, irrespective of task or node count, underscores MNIST's scalability owing to its inherent simplicity. Moreover, increasing tasks per node accelerates training by reducing convergence and execution times. Even as scaling extends to multiple nodes (N4 configurations), minor increases in execution time due to inter-node communication are offset by enhanced convergence rates, culminating in optimal model performance. Notably, the N4T8 setup attains a peak accuracy of 99%, albeit with a marginally extended execution duration.
The CIFAR-10 dataset, with its 60,000 32x32 colour images spanning ten categories, poses a more complicated problem. This complexity is reflected in slightly diminished accuracy rates and prolonged convergence and execution times. Single-node

configurations like N1T8, N1T16, and N1T24 witness accuracies ranging from 79% to 82%, with a slight dip as tasks increase. Nonetheless, akin to MNIST, augmenting tasks per node expedites training. However, scaling across multiple nodes introduces pronounced communication overheads, leading to increased execution times. For instance, the N4T24 configuration, entailing four nodes with 24 tasks each, experiences a significant execution time surge compared to single-node setups. Despite this, the model's pace in achieving its final accuracy accelerates with added nodes, even if overall training duration extends.
In conclusion, MNIST scales efficiently due to its simplicity, maintaining high accuracy and faster convergence with more resources. In contrast, CIFAR-10's complexity leads to trade-offs, where added nodes and tasks improve convergence but increase communication overhead and execution time, highlighting the need to balance computational power and efficiency.

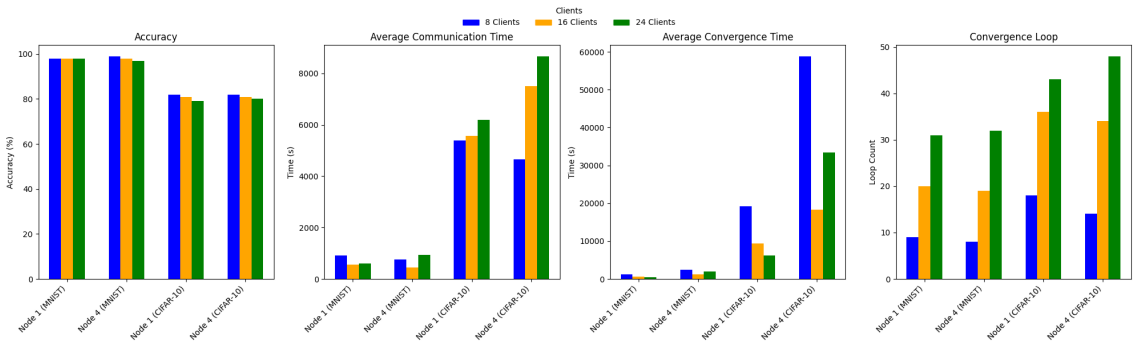

Figure 3.8: Scalability analysis of MNIST and CIFAR-10 datasets showing the impact of increasing tasks on accuracy, convergence time, communication time, and convergence loops. The plots compare the performance metrics across 8, 16, and 24 tasks for both datasets, highlighting the trade-offs between accuracy and computational resources

4

Conclusion and Discussion

4.1 Discussion

The findings in this thesis show how communication imperfections, like quantization and erasure, affect distributed optimization algorithms such as Federated Averaging (FedAvg) and Push-Pull. The study shows that the algorithms' effectiveness is greatly impacted by the extent of quantization and the likelihood of erasure, despite their ability to uphold accuracy and timely convergence.

For example, the studies show that decreasing the number of bits in quantization can help decrease communication costs, but it also adds noise that slightly reduces accuracy and leads to more convergence iterations. This can be seen clearly in the CIFAR-10 dataset, as the intricate data format increases the vulnerability of the model to quantization noise. Nevertheless, the trade-off is frequently offset by the advantages in communication effectiveness, particularly in situations with limited resources where minimizing transmitted data is essential. This is consistent with research, like the study by [32], which indicates that methods that prioritize efficient communication, such as quantization, are crucial for expanding federated learning systems, despite the possible effects on model accuracy.

Likewise, the effects of removal indicate that while these algorithms can handle a certain amount of data loss, there is a significant rise in convergence time and a small drop in accuracy, especially when dealing with more intricate datasets such as CIFAR-10. This implies that although FedAvg and Push-Pull can handle small communication failures well, their effectiveness decreases when there are frequent or severe losses of data. The findings show that Push-Pull, with its decentralized format, is somewhat more resistant to deletion than FedAvg, which heavily depends on consolidating updates at a central server. Push-Pull is a better option in situations with unreliable communication due to its resilience.

These findings underscore the need for adaptive strategies that can dynamically adjust quantization levels and employ robust error correction methods to mitigate the adverse effects of communication imperfections. For instance, implementing adaptive quantization schemes that adjust the bit rate based on network conditions, or integrating advanced error correction codes, could enhance the robustness of these algorithms without significantly increasing communication overhead. Future work could explore these adaptive techniques in greater depth, potentially combining them

with hybrid network topologies that optimize the trade-offs between communication efficiency, convergence speed, and model accuracy.

4.2 Conclusion

This thesis examined how different network structures and communication methods affect the effectiveness and expandability of distributed learning systems, specifically in the realm of Federated Learning. The results show that the structure of the network is crucial in deciding how well distributed learning systems perform, affecting both the time it takes to reach a solution and how efficiently they communicate. The Fully Connected (FC) and Random topologies showed high accuracy but were less efficient with longer convergence and communication times compared to the Star topology, which balanced efficiency and accuracy among the studied topologies.

Moreover, the study on scalability pointed out the obstacles in scaling up federated learning systems to support a larger client base. As the system grows, communication slowdowns become more noticeable, especially with intricate datasets such as CIFAR-10. While gradient quantization has displayed potential in lowering communication delays, more research is necessary to understand its impact in various situations. The necessity for adaptive strategies that can dynamically optimize accuracy, convergence time, and communication overhead becomes apparent as the system grows in size, highlighting the trade-offs involved.

The thesis also explored how distributed learning algorithms like FedAvg and Push-Pull are resilient to communication imperfections, such as quantization noise and data erasure. Although these algorithms stay strong to some extent, their effectiveness decreases in harsher conditions, especially when dealing with complex datasets. The decentralized structure of Push-Pull provides increased resilience, making it a better option for settings with unreliable communication.

Ultimately, the study highlights the significance of choosing suitable network structures and communication tactics to enhance the effectiveness of distributed learning systems. Adaptive techniques that can adjust to changing network conditions and data complexities will be crucial as these systems grow in size. Future research should concentrate on creating and combining these flexible approaches to improve the strength, effectiveness, and expandability of federated learning systems in various and demanding settings.

Bibliography

[1] T. Yang, X. Yi, J. Wu, *et al.*, "A survey of distributed optimization," *Annual Reviews in Control*, vol. 47, pp. 278–305, 2019.

[2] A. Nedi, A. Olshevsky, and M. G. Rabbat, "Network topology and communication-computation tradeoffs in decentralized optimization," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 953–976, 2018.

[3] J. Dean, G. Corrado, R. Monga, *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[5] J. Poushter *et al.*, "Smartphone ownership and internet usage continues to climb in emerging economies," *Pew research center*, vol. 22, no. 1, pp. 1–44, 2016.

[6] E. M. Noam and E. M. Noam, "Technology management in media and information firms," *Managing Media and Digital Organizations*, pp. 87–129, 2019.

[7] W. House, "Consumer data privacy in a networked world: A framework for protecting a privacy and promoting innovation in the globaeconom," *http://www.whitphi) nse pnY/siles/default/files/privac*, 2012.

[8] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[9] A. Nedi, A. Olshevsky, and C. A. Uribe, "Nonasymptotic convergence rates for cooperative learning over time-varying directed graphs," in *2015 American Control Conference (ACC)*, IEEE, 2015, pp. 5884–5889.

[10] A. Nedi and J. Liu, "Distributed optimization for control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 77–103, 2018.

[11] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trendső in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[12] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[13] A. Nedic, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017.

[14] C. Xi, V. S. Mai, R. Xin, E. H. Abed, and U. A. Khan, "Linear convergence in optimization over directed graphs with row-stochastic matrices," *IEEE Transactions on Automatic Control*, vol. 63, no. 10, pp. 3558–3565, 2018.

[15] A. Nedi and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 601–615, 2014.

[16] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the admm in decentralized consensus optimization," *IEEE Transactions on Signal Processing*, vol. 62, no. 7, pp. 1750–1761, 2014.

[17] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, "Optimal algorithms for smooth and strongly convex distributed optimization in networks," in *international conference on machine learning*, PMLR, 2017, pp. 3027–3036.

[18] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, IEEE, 2003, pp. 482–491.

[19] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *2012 ieee 51st ieee conference on decision and control (cdc)*, IEEE, 2012, pp. 5453–5458.

[20] J. Zeng and W. Yin, "Extrapush for convex smooth decentralized optimization over directed networks," *Journal of Computational Mathematics*, pp. 383–396, 2017.

[21] C. Xi and U. A. Khan, "Dextra: A fast algorithm for optimization over directed graphs," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 4980–4993, 2017.

[22] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, 2010, pp. 456–464.

[23] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of dnns with natural gradient and parameter averaging," *arXiv preprint arXiv:1410.7455*, 2014.

[24] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[25] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized consensus optimization with asynchrony and delays," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 293–307, 2017.

[26] S. Pu and A. Nedi, "Distributed stochastic gradient tracking methods," *Mathematical Programming*, vol. 187, no. 1, pp. 409–457, 2021.

[27] U.-E.-H. Alvi, W. Ahmed, M. Rehan, R. Ahmad, and A. Radwan, "A novel consensus-oriented distributed optimization scheme with convergence analysis for economic dispatch over directed communication graphs," *Soft Computing*, vol. 27, no. 20, pp. 14 721–14 733, 2023.

[28] A. Johannssen, N. Chukhrova, and Q. Zhu, *Symmetrical and asymmetrical distributions in statistics and data science*, 2023.

[29] S. Zheng, "Study of graph theory, distributed average consensus algorithm and centralized algorithm," *arXiv preprint arXiv:2101.10523*, 2021.

[30] Y. Yang, Z. Zhang, and Q. Yang, "Communication-efficient federated learning with binary neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3836–3850, 2021.

[31] M. Dahl, *Decentralized learning over wireless networks with imperfect and constrained communication: To broadcast, or not to broadcast, that is the question!* 2023.

[32] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Foundations and trendső in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[33] E. Darzi, N. M. Sijtsema, and P. van Ooijen, "A comparative study of federated learning methods for covid-19 detection," *Scientific Reports*, vol. 14, no. 1, p. 3944, 2024.

[34] S. Pu, W. Shi, J. Xu, and A. Nedi, "Push–pull gradient methods for distributed optimization in networks," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 1–16, 2020.

[35] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes," in *2015 54th IEEE Conference on Decision and Control (CDC)*, IEEE, 2015, pp. 2055–2060.

[36] L. Wei, Z. Ma, C. Yang, and Q. Yao, "Advances in the neural network quantization: A comprehensive review," *Applied Sciences*, vol. 14, no. 17, p. 7445, 2024.

[37] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[38] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *Fifteenth annual conference of the international speech communication association*, 2014.

[39] P. S. Bouzinis, P. D. Diamantoulakis, and G. K. Karagiannidis, "Wireless quantized federated learning: A joint computation and communication design," *IEEE Transactions on Communications*, 2023.

[40] A. I. Rikos, W. Jiang, T. Charalambous, and K. H. Johansson, "Distributed optimization with gradient descent and quantized communication," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 5900–5906, 2023.

[41] Y. Mao, Z. Zhao, G. Yan, *et al.*, "Communication-efficient federated learning with adaptive quantization," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1–26, 2022.

[42] K. Gupta, M. Fournarakis, M. Reisser, C. Louizos, and M. Nagel, "Quantization robust federated learning for efficient inference on heterogeneous devices," *arXiv preprint arXiv:2206.10844*, 2022.

[43] Y. Liu, S. Chang, and Y. Liu, "Quantization robust federated learning for efficient inference on heterogeneous devices," *IEEE Transactions on Big Data*, 2024. [Online]. Available: `https://openreview.net/forum?id=lvevdX6bxm`.

[44] M. Shirvanimoghaddam, A. Salari, Y. Gao, and A. Guha, "Federated learning with erroneous communication links," *IEEE communications letters*, vol. 26, no. 6, pp. 1293–1297, 2022.

[45] P. Zheng, Y. Zhu, Y. Hu, Z. Zhang, and A. Schmeink, "Federated learning in heterogeneous networks with unreliable communication," *IEEE Transactions on Wireless Communications*, 2023.

[46] P. Felber, A.-M. Kermarrec, L. Leonini, E. Riviere, and S. Voulgaris, "P ulp: An adaptive gossip-based dissemination protocol for multi-source message streams," *Peer-to-Peer networking and Applications*, vol. 5, pp. 74–91, 2012.

[47] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "Signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning*, PMLR, 2018, pp. 560–569.

[48] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns.," in *Interspeech*, Singapore, vol. 2014, 2014, pp. 1058–1062.

[49] H. Wu and P. Wang, "Node selection toward faster convergence for federated learning on non-iid data," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3099–3111, 2022.

[50] R. Xin and U. A. Khan, "A linear algorithm for optimization over directed graphs with geometric convergence," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 315–320, 2018.

[51] W. Li, T. Lv, W. Ni, J. Zhao, E. Hossain, and H. V. Poor, "Decentralized federated learning over imperfect communication channels," *IEEE Transactions on Communications*, 2024.

[52] L. Yu and J. Huang, "Cyclic federated learning method based on distribution information sharing and knowledge distillation for medical data," *Electronics*, vol. 11, no. 23, p. 4039, 2022.

[53] H.-C. Kim and Z. Ghahramani, "Bayesian classifier combination," in *Artificial Intelligence and Statistics*, PMLR, 2012, pp. 619–627.

[54] A. P. Dawid and A. M. Skene, "Maximum likelihood estimation of observer error-rates using the em algorithm," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 28, no. 1, pp. 20–28, 1979.

[55] M. T. Liu, "Distributed loop computer networks," in *Advances in computers*, vol. 17, Elsevier, 1978, pp. 163–221.

[56] J.-C. Bermond, F. Comellas, and D. F. Hsu, "Distributed loop computer-networks: A survey," *Journal of parallel and distributed computing*, vol. 24, no. 1, pp. 2–10, 1995.

[57] M. Jameel, J. Grabocka, M. u. I. Arif, and L. Schmidt-Thieme, "Ring-star: A sparse topology for faster model averaging in decentralized parallel sgd," in *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, Springer, 2020, pp. 333–341.

[58] M. Koibuchi, H. Matsutani, H. Amano, D. F. Hsu, and H. Casanova, "A case for random shortcut topologies for hpc interconnects," *ACM Sigarch Computer Architecture News*, vol. 40, no. 3, pp. 177–188, 2012.

[59] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive push-pull: Disseminating dynamic web data," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 265–274.

A

Appendix 1