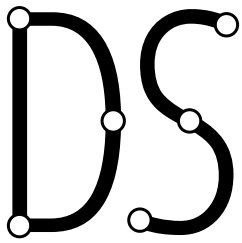
Master thesis

MatchCurv: A Communication-efficient Decentralized Federated Learning framework for Heterogeneous environment

Harsha Dussa

Kiel, Germany 2023



Kiel University
Department of Computer Science
Distributed Systems Group

Supervisor 1: M.Sc. Janek Haberer
Supervisor 2: Prof. Dr. Olaf Landsiedel

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 6. September 2023

Abstract

Federated learning (FL) has emerged as a promising approach to training machine learning models while maintaining data privacy. However, traditional centralized federated learning (CFL), with its reliance on a central entity, suffers from limitations such as a single point of failure and communication bottlenecks. To address these issues, decentralized federated learning (DFL) has been proposed. Despite its potential, finding a communication-efficient and robust DFL framework has been challenging, particularly in the presence of statistical and systems heterogeneity. This study aims to address the limitations of conventional DFL by introducing MatchCurv, a decentralized federated learning framework.

MatchCurv leverages FedCurv and Matcha to address statistical heterogeneity and improve communication efficiency, respectively. To handle systems heterogeneity, we accommodate partial work from stragglers. This approach results in increased resilience against non-identical data distributions, as well as network and computational limitations. Experiments show that models trained using MatchCurv outperform periodic decentralized stochastic gradient descent (PD-SGD) in terms of faster convergence and improved accuracy.

Keywords:
Machine Learing, Federated Learning, Decentralized Federated Learning, Matcha, FedCurv.

Contents

Chapter 1

Introduction

The widespread adoption of connected devices [1], [2] equipped with sensors has resulted in a vast amount of data that holds immense potential for advanced data analysis methods such as machine learning (ML). However, the decentralized nature of this data poses challenges [3] for collecting it in a central location for effective utilization. This task becomes even more complicated due to strict regulations [4] aimed at safeguarding the privacy of individuals, which can be compromised when sharing such data.

As sensor devices continue to advance in their computational capabilities, there is a growing appeal to conduct computations directly on the devices themselves. This approach ensures that the data remains on the device and that only the extracted insights are collected. However, a substantial hurdle remains in the coordination of collaborative data analysis across multiple devices, all while adhering to the constraint of not exchanging any data. These sensor devices, referred to as "edge devices" in this study, frequently exhibit heterogeneity in their communication and computational capabilities, posing a notable challenge when orchestrating the training process.

In 2016, Federated Learning (FL) was introduced by researchers at Google to address this challenge and showcase next-word prediction on smartphones [5]. In this instance, the FL approach allows edge devices, such as smartphones, to download the current next-word prediction model from the cloud and enhance it with the data stored on the device. The improvements are then summarized into a model update, encrypted, and transmitted to the cloud. In the cloud, the updates are combined into a consolidated model, which is then deployed for further training. Crucially, FL ensures that all training data remains on the edge devices and individual updates are not stored in the cloud, since even these updates can sometimes reveal some insights on the data distribution at the source.

In a broader sense, FL involves assigning each edge device the task of minimizing a parameterized objective function using its local data and a choice of solver. The parameters are subsequently shared with a central server, which aggregates them and then relays them back to the edge devices for further refinement. This iterative process continues until the desired results are achieved.

Federated learning has gained popularity across various domains, including smartphone user behavior analysis [6], [7], healthcare applications [8], and the Internet of Things (IoT) [9]. It enables privacy-aware applications to make use of distributed training data that would otherwise remain unusable, all while safeguarding user privacy and optimizing resource utilization. Some notable applications of FL encompass next-word prediction, face detection, and voice recognition on mobile devices; predictive healthcare within organizations like hospitals; and real-time adaptation in IoT devices such as wearables, autonomous vehicles, and smart homes [7], [8], [9].

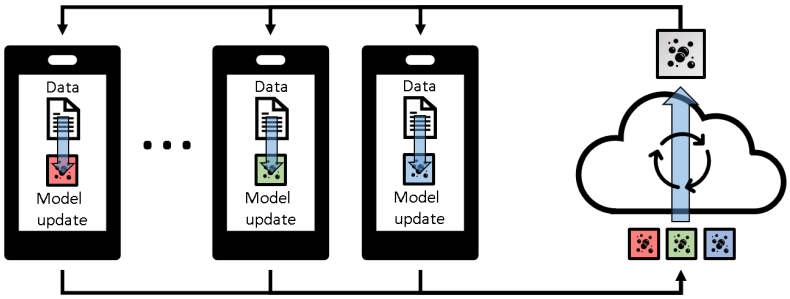


Figure 1.1: The figure provides a visual representation of the federated learning process, illustrating it with a specific example. In this scenario, individual smartphones are engaged in training a next-word prediction model using their locally available data. These smartphones then transmit the model updates to a central server. The server's role is to amalgamate these individual models into a new, consolidated model and subsequently distribute it back to the devices that participated in the training process. These devices, in turn, further enhance the new model by incorporating their own local data. This iterative process of training and model sharing continues until the desired level of model performance is attained. Ultimately, the trained model serves as the foundation for native next-word prediction on the devices.

To coordinate the training process, federated learning relies on a central server. However, placing trust in a central entity raises concerns regarding privacy, data security, and the potential for a single point of failure. Moreover, depending on a central server can lead to communication bottlenecks, limiting the scalability and efficiency of the system [10]. These challenges have generated a growing interest in decentralized paradigms, which aim to overcome these limitations and offer more robust and scalable solutions. In 2018, decentralized federated learning (DFL) [11] emerged as a novel approach. Unlike the traditional centralized model, DFL eliminates the need for a central entity. In this paradigm, edge devices independently learn statistical models using their local data and subsequently share the trained parameters directly with their immediate peers. The goal is to collaboratively reach a consensus on the parameters among the participating devices.
By removing the central entity, decentralized federated learning offers a more distributed and peer-to-peer approach to training models while alleviating the issues caused by using a centralized server. Nevertheless, several other challenges persist in federated learning, regardless of whether it is implemented in a centralized or decentralized manner. The most prominent of these challenges include:

- Communication Efficiency: The exchange of learned parameters between devices can be considerably slower compared to computation [12]. To mitigate this, it is crucial to minimize the size of shared parameters and reduce the frequency of their transmission, thus lowering overall communication costs.
- Statistical Heterogeneity: Data collected by individual devices often exhibits non-identical characteristics, violating the assumption of independent and identically distributed data. This heterogeneity poses difficulties in training a robust and generalizable model capable of effectively capturing variations across different devices [13].

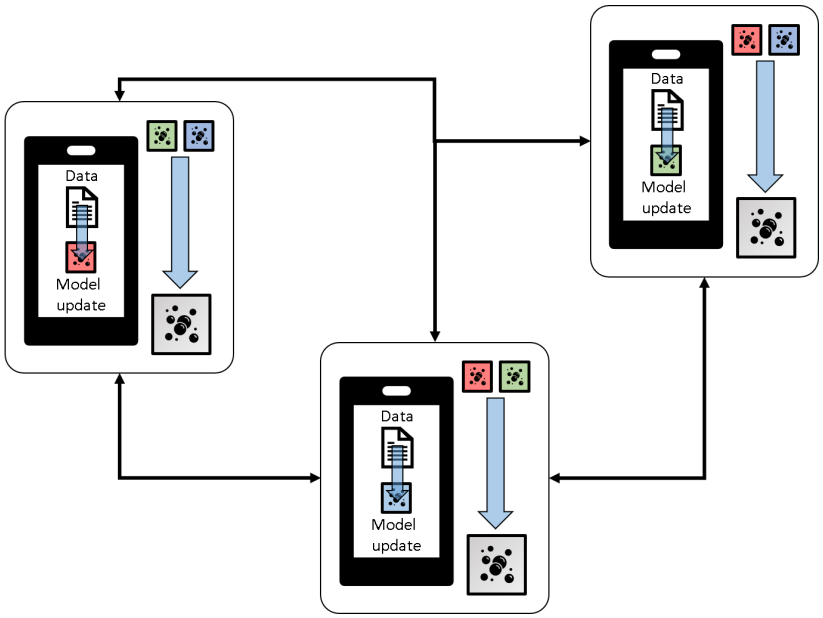


Figure 1.2: In DFL, the devices form an ad-hoc network and get model updates from their immediate neighbors (devices that are reachable in a single hop). The models are then aggregated by the devices themselves. The same iterative process of training and sharing model updates continues. But unlike centralized, federated learning, the devices only communicate with their peers.

- System Heterogeneity: Edge devices vary in terms of their computational power and networking capabilities. This diversity can lead to stragglers—devices that lag behind in the training process, thus slowing down overall progress. Effectively managing and optimizing the training process in the presence of systems heterogeneity is crucial for efficient federated learning [10].

Addressing these challenges necessitates continuous research to devise efficient communication protocols and effectively manage statistical and system heterogeneity within federated learning systems. While solutions to these problems have been well-explored in centralized federated learning (CFL), tackling all these issues simultaneously within a decentralized federated learning framework has proven to be challenging [14].
In this work, our goal is to overcome these challenges by introducing a novel framework that effectively handles statistical and system heterogeneity while enhancing communication efficiency. In this endeavour:

- To enhance communication efficiency, we leverage the Matcha framework [15] and adopt a strategy of more frequent parameter sharing with critical neighbors while reducing the frequency with others. This targeted approach optimizes the exchange of information (model updates) while minimizing overall communication costs.
- In addressing statistical heterogeneity, we integrate concepts from lifelong machine learning [16] and FedCurv [17]. We strategically penalize parameter changes in our framework to ensure that the trained model adapts to vary-

ing data distributions across devices, leading to improved generalization and performance.
- Furthermore, to tackle system heterogeneity and mitigate the impact of stragglers, we draw inspiration from FedProx [18] and implement a training timeout mechanism. This mechanism interrupts stragglers and accepts partial solutions, preventing them from impeding the overall training process.

We perform thorough experiments to validate the effectiveness of each of the proposed strategies, both individually and in combination, across various environments. The outcomes of our study reveal substantial improvements in the performance of the trained model when faced with statistical and system heterogeneity. Moreover, our approach demonstrates reduced communication demands compared to the traditional Periodic Decentralized Stochastic Gradient Descent (PD-SGD) [19], underscoring its efficiency in the context of decentralized federated learning.
With the introduction of our decentralized federated learning framework, our aim is to contribute to the advancement of decentralized learning techniques and address multiple challenges simultaneously. The remaining sections of this thesis are structured as follows:

In Chapter 2, we provide essential background information on federated learning and decentralized federated learning, emphasizing the specific challenges associated with FL. Chapter 3 presents a comprehensive review of related work in FL, highlighting the distinctions between this thesis and previous studies. Chapter 4 delves into the design of our DFL framework, called MatchCurv, offering a detailed explanation of strategies employed in orchestrating training among the decentralized federation of devices. Following the design chapter, Chapter 5 focuses on the implementation details of MatchCurv, describing the technical aspects and steps taken to realize our proposed framework with the help of UML diagrams. In Chapter 6, we present the evaluation results of our approach, analyzing the performance of MatchCurv compared to PD-SGD in various settings. Finally, in Chapter 7, we summarize our findings and conclusions about our framework and its effectiveness. We also discussed potential areas for improvement in MatchCurv and future work in the field of decentralized federated learning.

Chapter 2
Background

This chapter aims to provide the reader with the essential background knowledge necessary for understanding the design of our framework and the corresponding findings presented in this thesis, starting with a brief explanation of conventional machine learning (ML).

2.1 Machine Learning

In traditional machine learning paradigms, data is typically gathered from its source and transmitted to a central location for model training. This approach offloads the computationally intensive model training process from edge devices, which could include low-power sensor devices with limited capabilities for model training. However, this method raises concerns regarding the privacy of participants' data, as the central entity becomes susceptible to potential malicious actions or adversaries. Additionally, the transmission of substantial data volumes between participants and the central server introduces communication overhead.
Moreover, sophisticated data analysis demands models with billions of parameters, which, in turn, necessitate extensive training periods. In scenarios where data is inherently distributed or exceeds the capacity of individual machines, a centralized solution becomes impractical. To handle such datasets and expedite the training of machine learning models, distributed computing algorithms are employed. The ensemble of these algorithms constitutes distributed machine learning (DML) [20]. Within this framework, models are independently trained on multiple devices, with each participant forwarding model updates to a central server. The central server then aggregates these updates, often through averaging, to generate a unified model. After a predetermined number of communication cycles, convergence testing occurs on the central server to evaluate the progress of model training. This distributed methodology enables more efficient and scalable training, eliminating the need to transmit data while retaining the fundamental principles of conventional machine learning.

2.2 Federated Learning

The discourse surrounding distributed machine learning (DML) primarily focuses on harnessing the capabilities of distributed computing resources within data centers, where training devices share homogeneity in their computational and network capabilities. Additionally, the sharing of data samples among computing devices is often not restricted. Typically, the assumption is that data samples are distributed independently and exhibit identical distributions (IID) among the computing de-

vices. Consequently, there has been relatively limited research on handling non-IID data partitions in DML [13]. The existing research on distributed machine learning predominantly emphasizes reducing communication overhead.
Federated learning (FL), much like DML, revolves around training statistical models on individual edge devices in a distributed manner. In this context, an edge device is considered to be any device capable of participating in model training. The FL system comprises a federation of devices, each possessing its own local dataset. At the beginning of each training round, the server sends the current global state, such as the current model parameters, to these devices. Each device then trains the model using its local data and sends its update back to the server. The server subsequently incorporates these updates into its global state, integrating the contributions from each client. This process continues until a stopping condition is satisfied [21].
The pivotal distinction between federated learning and distributed machine learning lies in the assumptions made about the attributes of local datasets. Unlike DML, Federated Learning imposes strict limitations on sharing data samples and addresses the challenge of non-IID local datasets. It excels at handling heterogeneous datasets with diverse characteristics and sizes. Additionally, Federated Learning accommodates unreliable devices, such as smartphones and IoT devices, which are constrained by computational and communication limitations as well as limited battery power. In contrast, distributed learning typically relies on powerful data centers connected by high-speed networks, resulting in more reliable communication.
The ethos of federated learning builds upon the foundational concepts of DML and edge computing, extending their application to broader domains. FL is especially pertinent in situations where computing resources are unreliable, and label distributions are skewed. This decentralized approach aligns with the need for privacy-preserving machine learning applications in domains like healthcare and finance. Furthermore, Federated Learning taps into vast amounts of data from edge devices that would otherwise be inaccessible or impractical to transfer to a centralized server.

2.3 Decentralized Federated Learning

Traditionally, federated learning relies on a central server for coordinating the training process; hence, it is also known as centralized federated learning (CFL) [21]. However, this approach carries the risk of a single point of failure. Additionally, it can introduce communication bottlenecks, hindering scalability and system efficiency. To further illustrate the need for an alternative training paradigm, consider the previously introduced example of next-word prediction on mobile devices. In this example, multiple users of mobile devices want to collectively train a model. Currently, they must find and employ a trustworthy and capable central entity to orchestrate the training process. However, a more attractive solution is to leverage a decentralized approach, enabling users to train the model using their own resources. Decentralized Federated Learning (DFL) [11] addresses the concerns associated with centralized federated learning by empowering individual devices to initiate the training and make decisions regarding the model type, loss function, and optimization algorithm. Other devices can then choose to participate in the training process. During training, each device collects parameter updates from its immediate neigh-

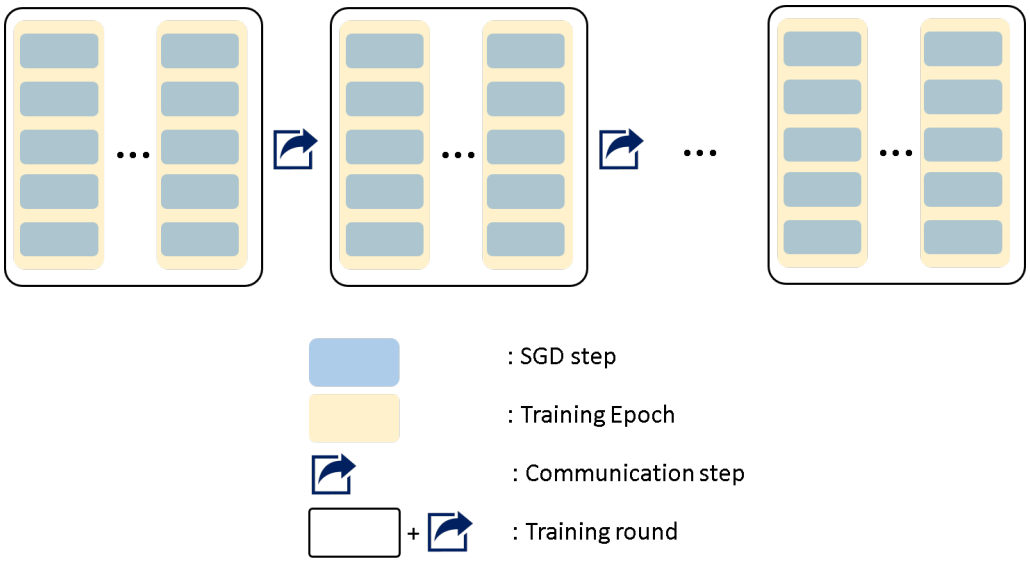


Figure 2.1: The figure illustrates the various steps in training a deep neural network through federated learning. A typical training round consists of two main components: a training step and a communication step. During the training step, the model undergoes training for multiple epochs, and within each epoch, there are numerous mini-batch updates. Once the training step is completed, the model updates are either transmitted to a central entity in the case of centralized federated learning or broadcast to peers in the case of decentralized federated learning.

bors and aggregates them locally. DFL also delegated server responsibilities, such as synchronizing training rounds, to edge devices. This can be achieved through training deadlines or, alternatively, by embracing an asynchronous training model. DFL could be more appealing in scenarios like the training of ML models among hospitals or institutions where employing a central entity is difficult. While DFL with a fully connected network resembles CFL, DFL allows for more flexible network topologies where devices only exchange updates with their immediate neighbors, reducing communication overhead and potentially improving scalability in large-scale federated learning settings.

2.4 MATCHA

While federated learning mitigates issues arising from conventional machine learning in privacy-sensitive scenarios, it introduces several challenges of its own. Unlike CFL, where devices only need to communicate with the central server, devices involved in DFL training must establish communication with all of their peers. This becomes particularly challenging in networks with a higher average nodal degree. Additionally, when devices have varying numbers of neighbors, it results in different communication overheads within the network, leading to stragglers. One way to address this issue is for devices to opt to communicate with only a single neighbor at a time or a subset of neighbors. This can be achieved by cyclically engaging with neighbors or selecting a random subset at each training round.

However, choosing to communicate with only a subset of neighbors to reduce communication costs can potentially worsen information dissemination. To effectively manage this challenge and expedite the spread of information, it becomes imperative to strategically select neighbors for sharing model updates. In a fully connected network, devices are indistinguishable from a topological perspective. Conversely, in an ad-hoc network, certain devices might hold higher topological significance and should receive model updates more frequently than others. To illustrate this, consider a scenario where a device bridges two clusters of devices. In such instances, the bridging device should receive updates from both clusters more frequently than the devices within each cluster. Consequently, the probability of selecting these pivotal devices should be higher compared to others.
To understand the topological importance of devices in a DFL network, we need to establish a few key points. In decentralized federated learning, the federation of devices participating in the training form an ad-hoc network, which can be represented as an undirected, unweighted graph $G(V, E)$, where V is the set of devices, each represented by a unique identifier (UID), and $E \subseteq V \times V$ is the set of device UID pairs representing network connections between the devices in the pair. With this graph representation, we can explore graph metrics that correlate with better graph connectivity and information dissemination. One such metric from spectral graph theory is algebraic connectivity [22]. For a graph represented by its adjacency matrix, the algebraic connectivity is defined as the second-smallest eigenvalue of the Laplacian matrix. Where,

Definition 1 (Laplacian Matrix of a network) *The Laplacian matrix $L(G) = [L(i,j)]$, $1 \leq i, j \leq n$ of network $G(V, E)$ with $|V| = n$ is an $n \times n$ matrix, with:*

$$L(i,j) = \begin{cases} deg(v_i) & \text{if } i = j \\ -1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Here, $i, j \in \{1, \ldots, n\}$ are node indices, while v_i stands for the ith node. Alternatively,

$$L(G) = D(G) - A(G) \tag{2.2}$$

where $A(G)$ the $n \times n$ adjacency matrix, $D(G)$ the $n \times n$ diagonal matrix of the network G with the (i,i)-th entry as $deg(v_i)$, representing the count of neighbors.

Definition 2 (Algebraic Connectivity) *The algebraic connectivity λ_2 of a (connected) graph is defined as the second smallest eigenvalue (λ_2) of the Laplacian matrix of a graph with n vertices. This parameter serves as a comprehensive gauge of "graph connectivity quality" with values in range $[0, n]$. This eigenvalue is greater than 0 if and only if graph G is connected and the further it deviates from zero, the more difficult it becomes to separate the graph into independent components.*

Algebraic connectivity provides a more informative measure of how well the graph is connected compared to metrics like the average nodal degree [23]. Increasing the algebraic connectivity of the network has been observed to correlate with a decrease

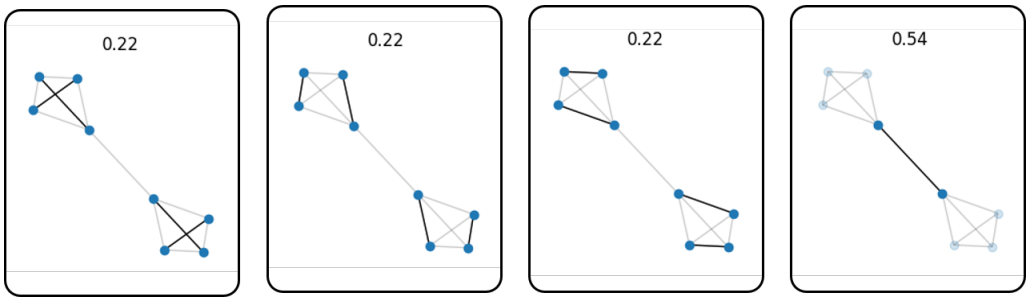


Figure 2.2: In the above figures, we can see different matchings (opaque) of a barbell graph (transparent). Each matching is labeled with its corresponding activation probability. Notably, the final matching, which includes the bridging edge, holds a higher probability in contrast to the other matchings despite having a smaller number of edges.

in the characteristic path length. A lower characteristic path length signifies the presence of shorter paths between devices, facilitating faster communication and information propagation. Therefore, when selecting a neighbor subset, we can do so strategically to improve algebraic connectivity and, in turn, reduce the impact on information dissemination caused by selecting a smaller subset.
In the Matcha framework [15], a communication budget $C_b \in [0, 1]$ is introduced, representing the proportion of parameter shares over the links in the network throughout the training. For example, by setting $C_b = 0.1$, Matcha achieves a $10\times$ reduction in communications compared to a full $C_b = 1$. Furthermore, the graph representing the network of devices is divided into m subgraphs known as matchings. Each matching is assigned a probability that reflects its contribution to the overall graph's connectivity. By assigning higher probabilities to matchings that contribute to higher algebraic connectivity, communication along these edges is prioritized, optimizing the network topology for improved communication efficiency. After the devices finish their individual training steps, the matchings are chosen randomly based on the probabilities assigned to them, and the devices within the chosen matchings share their model updates along the corresponding edges.

Definition 3 (Matching) *A matching in a graph $G(V, E)$ is a subset of the edges E in which no two edges share a common vertex. In other words, a matching is a set of edges without any common endpoints. A maximum matching is a matching of the largest possible size within a given graph.*

The probabilities for the matchings $P = \{p_1, ..., p_m\}$ are calculated by solving the optimization problem:

$$\max_P g(P) := \lambda_2 \left(\sum_{j=1}^{m} p_j L_j \right) \tag{2.3}$$

$$\sum_{j=1}^{m} p_j \leq C_b.m$$

$$0 \leq p_j \leq 1, \forall j \in \{1, 2, ..., m\}$$

Here, p_j represents the probability assigned to the j^{th} matching and L_j represents its Laplacian matrix. By incorporating the communication budget and assigning probabilities to matchings based on their contribution to graph connectivity, the Matcha framework optimizes the network topology in DFL, allowing for better information dissemination across the network. This results in more efficient utilization of network resources, making it more practical and feasible in real-world scenarios with limited communication capabilities.

2.5 FedCurv

With Matcha, we ensure the efficient utilization of network resources. To address the next problem we alluded to, we incorporated an existing CFL framework designed to handle statistical heterogeneity called FedCurv [17]. To explain FedCurv, we will begin by describing the optimization problem implicit in federated learning, which is referred to as federated optimization. This will help establish a connection to distributed optimization. In federated optimization, the typical objective is to minimize the following function:

$$\min_{w} F(w) = \sum_{k=1}^{n} p_k F_k(w) \tag{2.4}$$

Here, n represents the total number of devices. The term F_k represents the local objective function specific to the k^{th} device, and w represents the parameters of the function. To determine the relative impact of each device, one can assign the p_k values accordingly. The participating devices use their local solver to find the optimal parameters for the local objective function, and these parameters are gathered and combined by a central entity. The aggregated parameters are then shared back with the participating devices, enabling them to continue refining the solution. Through this iterative approach, the aim is to identify the optimal parameters that collectively minimize the objective function across all edge devices.
There are several frameworks proposed to facilitate this process. In a naïve application of stochastic gradient descent (SGD) to federated optimization, as shown in equation (2.4), a single batch gradient calculation is performed prior to each communication round [24]. This approach, referred to as FedSGD, is computationally efficient. However, it often requires a substantial number of training rounds and, consequently, a significant amount of communication to achieve satisfactory model performance. The federated averaging algorithm addresses this concern by introducing the concept of performing multiple SGD steps before each communication round. In this scheme, each client engages in numerous gradient descent steps spanning multiple epochs using its local data and the current model.
In FedSGD, for each training round t, every client k computes gradients θ_k for parameters w_t that minimize their respective local objective function $F_k(w_t)$. The central server then aggregates these gradients and applies the update $w_{t+1} = w_t - \eta \frac{1}{n} \sum_{k=1}^{n} \theta_k$, where n represents the total number of clients and η is the learning rate. Alternatively, this update can be individually executed for each client as $w_{k,t+1} = w_t - \eta\theta_k$, followed by the combination of these client models into a composite model

$w_{t+1} = \frac{1}{n} \sum_{k=1}^{n} w_{k,t+1}$. By adopting this approach, additional computation can be incorporated into each client's process by performing the local update iteratively multiple times before the aggregation phase. This extended procedure is known as Federated Averaging (FedAvg) [5]. The amount of computation performed between communication rounds is determined by the number of epochs E in each training round and the size of the minibatch B. Furthermore, setting $E = 1$ with $B = \infty$ corresponds to the FedSGD approach.

While FedAvg has demonstrated empirical success in handling heterogeneous settings, it falls short of fully addressing the challenges associated with statistical heterogeneity and lacks theoretical convergence guarantees in such scenarios [18]. Statistical heterogeneity refers to the way devices often generate and collect data in a non-identically distributed manner, leading to differences in the underlying data distributions across devices. For example, in the previously described next-word prediction task, mobile phone users may exhibit diverse language use patterns, and the number of data points collected by each device can vary significantly. When data samples among devices are independently and identically distributed (IID), we categorize this situation as the IID setting. In cases where this assumption does not hold, we refer to it as the Non-IID setting.

This departure from the standard paradigm of independent and identically distributed data contradicts a widely accepted assumption in distributed optimization. Consequently, during the multiple local updates performed between communication steps, there's a possibility that the local model becomes more aligned with the local data and forgets the previously learned global model. This divergence among models on different devices occurs because the local data distribution does not accurately represent the overall data distribution in a non-IID environment. To overcome these limitations, FedProx [18] was introduced as a generalization and re-parametrization of FedAvg.

To prevent excessive local updates from causing model divergence due to non-IID data distribution, FedProx introduces a proximal term to the local sub-problem. Instead of solely minimizing the local function, each device aims to minimize the modified objective using an inexact solver:

$$\min_{w} \bar{F}_k(w; w_{t-1}) = F_k(w) + \frac{\mu}{2} \|w - w_{t-1}\|^2 \tag{2.5}$$

In this context, w_{t-1} represents the global state or parameters from the previous round (t-1). The proximal term $\frac{\mu}{2} \|w - w_{t-1}\|^2$ addresses statistical heterogeneity by constraining local updates to remain close to the initial global model. It also accommodates system heterogeneity by allowing the safe incorporation of varying amounts of local work. It's important to note that FedAvg serves as a special case of FedProx, where the proximal coefficient μ equals 0, and no consideration for system heterogeneity.

However, introducing a penalty like the proximal term in FedProx, which captures the change in parameters as a whole, may inadvertently impede the training process, especially when the proximal coefficient is not chosen carefully. To provide a more effective solution, we reach for FedCurv [17], an adaptation of the Elastic Weight Consolidation (EWC) algorithm tailored specifically for federated learning.

2.5.1 Elastic Weight Consolidation

EWC originates from the realm of lifelong machine learning [16], which deals with the challenge of sequentially learning multiple tasks (e.g., task A and task B) using the same model without forgetting the knowledge acquired from previously learned tasks. Essentially, the goal is to learn new tasks while maintaining satisfactory performance on earlier tasks. In a similar vein, Federated Learning aims to enhance the model without disturbing the global model trained in previous rounds.
To combat the problem of forgetting, some proposed solutions involve creating a small subset of data that is shared among all edge devices. However, this approach contradicts the data privacy objectives of Federated Learning, as it necessitates sending data between devices or to a central point.
In contrast, Elastic Weight Consolidation (EWC) takes a different approach compared to sharing data or indiscriminately penalizing parameter changes, as seen in FedProx. Instead, it focuses on identifying the parameters most relevant to task A. During the learning process of task B, EWC penalizes any modifications made to these specific parameters. The core idea underlying this technique is rooted in the over-parameterization of deep neural networks. This over-parameterization enables networks to find an optimal solution for task B within the vicinity of previously learned parameters for task A. This selective penalization is achieved by utilizing the diagonal elements of the Fisher Information Matrix (FIM) to monitor and control deviations in the parameter vector that may lead to forgetting. The optimization problem for EWC is as follows:

$$\min_{w} F(w) = F_B(w) + \lambda(w - w_A)^T \mathrm{diag}(I_A)(w - w_A) \tag{2.6}$$

Here, F_B represents the objective function for task B, w_A denotes the parameters from task A, I_A is the Fisher Information Matrix computed from the model trained for task A, and λ represents the EWC constant.

2.5.2 Fisher Information Matrix

In order to determine the important weights for a task, we can approach the parameterized optimization problem from a Bayesian perspective. The search for optimal parameters can be framed as an attempt to find the most probable values for them, given the training data D. Assuming that the data can be divided into two independent parts, D_A, and D_B, we can employ Bayes' rule to compute the conditional probability as follows:

$$\log p(w|D) = \log p(D_B|w) + \log p(w|D_A) - \log p(D_B) \tag{2.7}$$

The Fisher Information Matrix (FIM) serves as an approximation for the posterior probability distribution of the weights for task A, assuming it conforms to a Gaussian distribution. The FIM is a square matrix whose elements are calculated using the second-order derivatives of the log-likelihood function with respect to the parameters. It captures the curvature of the likelihood function and offers valuable insights into the sensitivity of the model's predictions to changes in the weights. By focusing on the diagonal of the FIM, the approximation assumes that the weights are

independent of each other, simplifying the distribution into a multivariate Gaussian with diagonal covariance [25].
Indeed, based on this information, we can grasp how FedCurv is an adaptation of the Elastic Weight Consolidation (EWC) algorithm, finely tuned for the domain of federated learning. It's akin to considering the individual models from previous training rounds as distinct tasks and penalizing the model for deviating from the parameters of other devices from the prior round, with their respective importance quantified by their FIMs. The optimization problem for federated learning with FedCurv can be expressed as:

$$\min_{w} \hat{F}_{t,k}(w) = F_k(w) + \lambda \sum_{j \in K \setminus k} (w - w_{t-1,j})^T \mathrm{diag}(I_{t-1,j})(w - w_{t-1,j}) \tag{2.8}$$

Here, K represents the set of devices, and $w_{t-1,j}$ denotes the model parameters from device j in the $(t-1)$th training round. Similarly, $I_{t-1,j}$ corresponds to the Fisher information from device j in the $(t-1)$th round.
Moreover, in order to preserve network bandwidth and facilitate scalability, this equation could be restructured, eliminating the need for devices to exchange parameters and FIMs with every other device in the network. Instead, when the central entity receives these parameters and FIMs, it can calculate the aggregate penalties for each device and share only those penalties. So the restructured equation becomes:

$$\min_{w} \hat{F}_{t,k}(w) = F_k(w) + \lambda w^T u_{t-1} w - 2\lambda w^T v_{t-1} + \mathrm{const} \tag{2.9}$$

With,

$$u_{t-1} = \sum_{j \in K} \mathrm{diag}(I_{t-1,j}) \tag{2.10}$$

$$v_{t-1} = \sum_{j \in K} \mathrm{diag}(I_{t-1,j}) w_{t-1,j} \tag{2.11}$$

Now, the devices only need u_{t-1} and v_{t-1} values to incorporate into their penalties, which the central entity promptly calculates and shares at each training round.

In their evaluations, FedCurv showcases better performance compared to FedAvg and FedProx by harnessing the Fisher Information Matrix to selectively penalize parameter changes crucial for data or tasks from other devices. However implementing FedCurv requires a central entity, or the devices have to be connected with every other device directly. So, fully adapting it to DFL with an ad-hoc network structure becomes tricky. Therefore, in this work, we are content with FIMs from the immediate neighbors of a device.

Chapter 3
Related Work

In previous sections, we discussed foundational concepts in federated learning and machine learning. Now, we introduce related works in federated learning and emphasize the distinctive features of our framework compared to existing approaches.

3.1 Decentralized Federated Learning Frameworks

Decentralized Federated Learning (DFL) draws inspiration from decentralized optimization [26] and decentralized stochastic gradient descent techniques [27]. The field of decentralized optimization has explored various methods for solving convex optimization problems, including the Alternating Direction Method of Multipliers (ADMM) [28], Dual Averaging [29], Byzantine-resilient Distributed Coordinate Descent [30], and Newton Tracking [31]. However, the introduction of decentralized SGD [32] by Liu et al. in 2009 marked a significant milestone in DFL's development, demonstrating sublinear convergence to the optimal solution. Subsequent research has improved the convergence rate and performance of decentralized gradient descent (DGD). For example, Yuan et al. (2016) achieved linear convergence of DGD to a neighborhood of the global optimum using a constant step size for strongly convex objectives, while Shi et al. (2015) proposed the EXTRA algorithm, which enhances performance compared to DGD through a gradient tracking technique [32]. It's worth noting that research in DFL has extended to address non-smooth and non-strongly convex objectives frequently encountered in machine learning applications. The CoLa algorithm [11], introduced in 2018, specifically targets such objectives and outperforms other methods like DIGing and decentralized ADMM (DADMM) [33], [34] in empirical experiments. This work guarantees sublinear convergence rates for general convex objectives without assuming IID data distributions, unlike other stochastic methods.
Additionally, various frameworks have emerged to address the unique challenges of DFL in specific applications. For example, BrainTorrent [35] focuses on medical applications, establishing a dynamic peer-to-peer environment where centers interact directly, eliminating the need for a central authority. Autonomous blockchain-based federated learning (BFL) [36] is designed for privacy-aware and efficient vehicular communication networking, enabling on-vehicle machine learning without centralized training data or coordination by exchanging and verifying local on-vehicle machine learning (oVML) model updates in a distributed manner, leveraging blockchain's consensus mechanism.

In summary, decentralized optimization has garnered significant attention in the machine learning community, leading to advancements that address challenges related to network topologies, non-smooth objectives, and distributed scenarios. While

progress has been made in various areas of DFL, further research is needed to explore its potential in different machine-learning applications. Although several frameworks have been introduced for training in the CFL, the quest for a robust DFL framework that effectively tackles its prominent challenges remains ongoing. In the following sections, we demonstrate how specific DFL frameworks address distinct challenges and emphasize our approach's comprehensive solutions to these issues, aiming to introduce a framework that effectively addresses the most prominent challenges in DFL.

3.2 Communication-Efficient DFL

To reduce the high communication costs of decentralized federated learning, researchers have explored various strategies. One approach is compressing the exchanged parameters, as seen in methods like QSGD [37], [38], which involve nodes exchanging quantized versions of their local models. Provided that quantization doesn't affect model performance, it can significantly reduce parameter size, facilitating faster communication in network-constrained environments.
Compression methods can be combined with other communication reduction techniques, such as optimizing parameter update frequencies. The Federated Averaging algorithm (FedAvg) [5], introduced by McMahan et al., is notable in this regard. FedAvg reduces parameter update frequency by aggregating parameters after several epochs, effectively reducing communication overhead. Building on this concept, researchers have adapted FedAvg to the decentralized setting, introducing decentralized FedAvg with momentum (DFedAvgM) [39]. In DFedAvgM, each client conducts stochastic gradient descent with momentum and communicates solely with neighboring clients.
Beyond optimizing update frequency, researchers have introduced innovative methods to enhance communication efficiency in DFL. For instance, the Segmented Gossip Aggregation method [40], introduced by Chenghao Hu et al., distributes the transmission task across multiple links through segmented pulling. Instead of exchanging the entire model parameters with other workers, segmented pulling enables a worker to retrieve different parts of the model parameters from various workers and reconstruct a mixed model for aggregation, effectively reducing communication overhead.
In DFL, the topology of training devices can be leveraged to form effective communication protocols. One such approach is the Ring-allreduce method introduced in Baidu, which organizes workers in a ring structure. This method allows each worker to send gradients to the next worker in a clockwise direction and receive gradients from the previous worker, thus reducing communication complexity with linear growth. Similarly, tree and graph topologies have been proposed to minimize communication costs, though they may involve multiple hops between workers, potentially slowing convergence due to increased latency.
Another notable approach is NetMax [21], which focuses on optimizing peer selection for communication efficiency. In NetMax, each worker node selects a peer randomly during each iteration, considering the availability of high-speed links between workers. Peers with high-speed links are chosen with a higher probability, facilitating

faster and more efficient communication. By leveraging network characteristics and optimizing the probability distribution for peer selection, NetMax aims to minimize total convergence time and accelerate the training process in DFL.
However, heuristics like link speed only capture the local importance of a neighbor, and sharing model updates with nodes having high-speed links might not always result in faster information dissemination. To effectively leverage topology information, Matcha [15] proposes an algorithm that balances error convergence speed and runtime in DFL. Critical links within the network are identified, and communication is prioritized more frequently over these links, while less critical links are utilized less frequently, conserving communication time and resources.

In this study, we adhere to the conventional practice of performing multiple local updates before communication in decentralized federated learning. However, instead of employing random neighbor selection or local heuristics like connection speed, we utilize the concept of algebraic connectivity to assign importance to neighbors and determine their selection accordingly using Matcha. It's worth noting that the Matcha framework does not explicitly explore scenarios such as non-IID data and system heterogeneity in its evaluation. In this work, we aim to broaden Matcha's applicability by combining its concepts with other solutions designed to address different challenges in DFL. By integrating ideas from various approaches, we seek to measure the effectiveness of this combined approach in mitigating the challenges faced in DFL.

3.3 Handling Statistical Heterogeneity in DFL

Addressing non-IID data distribution has been a contemporary concern in federated learning since its inception. While the FedAvg framework has shown empirical promise in handling statistical heterogeneity, it poses challenges for rigorous analysis due to its local updating scheme.
Numerous studies have explored convergence guarantees and strategies for handling statistical heterogeneity in federated learning. However, some of these investigations assume that all devices participate in each round of communication, which may not be practical in realistic federated networks. Additionally, these approaches often rely on specific solvers like stochastic gradient descent or gradient descent (GD) and introduce additional assumptions such as convexity or uniformly bounded gradients [18]. Alternative solutions like FedProx [18] have emerged in CFL research, avoiding such assumptions. FedProx proposes the use of a proximal term and a local inexact solver, offering both theoretical and empirical convergence guarantees.
Drawing inspiration from closely related fields like multitask learning, some researchers advocate training individual models for each device rather than a single global model across the network [41]. This approach enables the creation of personalized models tailored to the unique characteristics of each device's data. In a similar vein, another approach employs hierarchical clustering [42] to group clients based on the similarity of their local updates. By dividing clients into clusters, it becomes feasible to train them independently and concurrently on specialized models

customized to their specific data distributions.
From the realm of lifelong learning [16], FedCurv [17] introduces the concept of elastic weight consolidation to Federated Learning. Empirical evidence suggests that FedCurv outperforms FedAvg and FedProx in the face of statistical heterogeneity, though it does not make claims about system heterogeneity, unlike FedProx.
Heuristic approaches, such as sharing local device data or server-side proxy data, have also been proposed to tackle statistical heterogeneity in FL [43]. However, sending local data to the server compromises privacy, while distributing globally shared proxy data to all devices necessitates substantial effort to generate or collect auxiliary data.
In the context of decentralized federated learning, Onoszko et al. introduced a synchronous method called performance-based neighbor selection (PENS) to address statistical heterogeneity [44]. PENS selects nodes with similar data distributions for communication. With a similar ethos, FedHP [45] focuses on controlling local updating frequency along with the network topology. It dynamically adjusts updating frequencies for heterogeneous nodes, allowing them to adapt their training processes based on their respective data distributions.

In our research, we adapted FedCurv for decentralized federated learning, specifically restricting the use of Fisher Information Matrices to immediate neighbors. It's noteworthy that, to our knowledge, FedCurv has not been adapted and evaluated within a DFL framework. Therefore, our objective with this study is to demonstrate its effectiveness in the context of DFL, considering both statistical and system heterogeneity.

3.4 Handling System Heterogeneity in DFL

System heterogeneity encompasses variations in devices' computational and communication capabilities due to differences in hardware and network connectivity. This diversity can lead to stragglers—devices that lag behind during training.
Traditional federated learning methods, like FedAvg [5], do not consider these underlying system constraints. They typically drop devices that fail to complete a set number of local updates within a specific time window or selectively sample a subset of devices based on their system capabilities. However, this approach can hinder convergence by limiting the effective devices contributing to training and introducing bias if dropped or sampled devices possess specific data characteristics [18].
While frameworks like FedProx [18] accept partial work from stragglers, both dropping stragglers (as in FedAvg) and naively incorporating partial information (as in FedProx) can negatively impact convergence in centralized federated learning. FedProx addresses this by introducing a regularization term to enhance method stability.
In contrast to assuming identical local update frequencies and fixed neighbors for all edge nodes, the FedHP [45] framework explores adaptively assigning varying local update frequencies for heterogeneous nodes and adjusting the network topology to eliminate idle time caused by synchronization.

Another framework, HADFL [46], supports decentralized asynchronous training on heterogeneous devices. In HADFL, devices train models locally using heterogeneity-aware local steps with their respective data. In each aggregation cycle, devices are probabilistically selected to perform model aggregation. Approaches like HeteroFL [47] challenge the assumption that local models must adhere to the same architecture as the global model. Instead, they allocate models of different sizes (e.g., varying units or layers in a neural network) to devices based on their individual capabilities.

In our work, we advocate for accommodating stragglers rather than dropping them. Stragglers can be topologically important, acting as bridges connecting distinct clusters of devices. Removing such devices from training disrupts the flow of model updates between these clusters. Instead of imposing a fixed number of local updates, we propose the adoption of a time-based deadline for the training step. This introduces flexibility to accommodate stragglers and mitigate synchronization challenges without a central entity. This approach ensures that all devices' contributions, including potentially important stragglers in terms of network topology, are integrated into the federated learning process. Consequently, it promotes improved convergence and overall performance in decentralized settings. Our objective is to combine approaches from centralized federated learning and distributed machine learning and adapt them to the context of centralized federated learning. In doing so, we ensure that the chosen methods are compatible and do not conflict with each other. To achieve this, we have selected two frameworks from existing research: Matcha and FedCurv. Each of these frameworks brings unique contributions to the DFL setting. Matcha aids in effective information dissemination, and FedCurv introduces adaptive parameter penalization, mitigating divergence from the global model. By integrating these frameworks, we aim to leverage their respective strengths and create a comprehensive solution that effectively addresses the challenges in DFL.

Chapter 4
Design

This chapter explores the design intricacies and pivotal decisions behind the development of our DFL framework, MatchCurv. The framework consists of several essential stages: consensus, Matcha, model training, and communication. Each of these stages plays a crucial role in the framework's functionality and effectiveness. In the following sections, we will provide detailed insights into each step.

4.1 MatchCurv and its Components

In contrast to the CFL approach, our DFL framework empowers inactive edge devices to independently initiate model training. The process begins when a device proactively chooses to start model training. This device has the autonomy to set hyperparameters for the training process. Other devices that receive an invitation to participate in the training can decide whether to join this collaborative learning effort. The invitation includes chosen hyperparameters, giving participating devices insight into critical training details such as the number of epochs (training time in our case), batch size, and rounds. These parameters help devices make informed decisions about participation. If a device finds the model size too demanding or the number of training rounds excessive, it can decline the invitation. If a device opts to participate, it shares the training invitation and hyperparameters with its neighboring devices, who follow the same procedure, spreading the information across the network. Once devices receive responses from their neighbors indicating their decision to participate, they incorporate their unique identifiers (such as IP addresses) into their neighbor sets. This set forms the initial subgraph of the complete training network. Following this, devices initiate the consensus steps.
The first crucial consensus step, detailed in subsequent sections, enables devices to agree on the structure of the training network or graph. Initially, the subgraph consists of only the immediate neighbors of a device. Through the consensus step, this subgraph progressively expands until all received subgraphs from neighboring devices achieve uniformity. This step is paramount as it establishes the foundation for the subsequent Matcha process. In Matcha, the fully realized network topology is deconstructed into individual matchings, with each of the probabilities assigned based on their significance within the network.
The second essential consensus step in the algorithm revolves around agreeing on the starting time for the training process among all participating devices. Each device proposes a starting time, and the latest among these proposals is adopted as the starting time. This synchronization promotes a collaborative and well-coordinated training environment. Leveraging the assumption of a synchronized global clock, all devices initiate the training loop simultaneously, ensuring a uniform and coordinated start.

Algorithm 1: MatchCurv Algorithm

```
/* - Device decides it wants to train.                    */
/* - Device picks hyperparameters.                        */
Broadcast Invite and hyperparameters.
Wait for replies.
N := {v|v ∈ Responding Device UIDs}
Perform Consensus step for Topology (Algorithm 2).
Perform Matcha (Algorithm 3).
Perform Consensus step for Start time (Algorithm 2).
Wait until set time.
foreach round ∈ [1, r] do
    Perform local updates (Algorithm 5).
    Compute FIM (Algorithm 6).
    Test the model.
    activations := Get Activations (Algorithm 4).
    foreach neighbor ∈ activations do
        Send Model updates and FIM to neighbor.
    Wait for replies until timeout.
    Merge models.
    Save FIMS.
Save Results.
```

At the algorithm's core, there's an iterative training loop spanning a predetermined number of rounds. In each iteration, participating devices perform localized updates to their model parameters using their unique datasets. To address potential delays caused by slower devices, we enforce a training timeout to manage the risk of straggler issues. This timeout approach replaces a fixed number of epochs, allowing different devices to train varying numbers of epochs within the timeframe, accommodating workload variations.

After these personalized updates, devices assess their models' performance against predefined benchmarks. The next step in the loop involves computing the Fisher Information Matrix (FIM) and activations, facilitated by the introduced activations algorithm. Activations refer to neighboring devices selected for exchanging model updates and computed FIMs. This exchange follows a well-defined synchronization protocol, ensuring communication aligns with the overarching training timeline. This approach minimizes delays and ensures smooth progress. Following the timeout period, individual models are merged into a consolidated model, and the corresponding FIMs and model updates are retained for calculating penalties in the next training iteration.

Upon successfully completing the specified number of training rounds, the algorithm concludes its execution by recording comprehensive outcomes. These outcomes include updated model parameters, performance metrics, and insights gained from the collaborative learning journey.

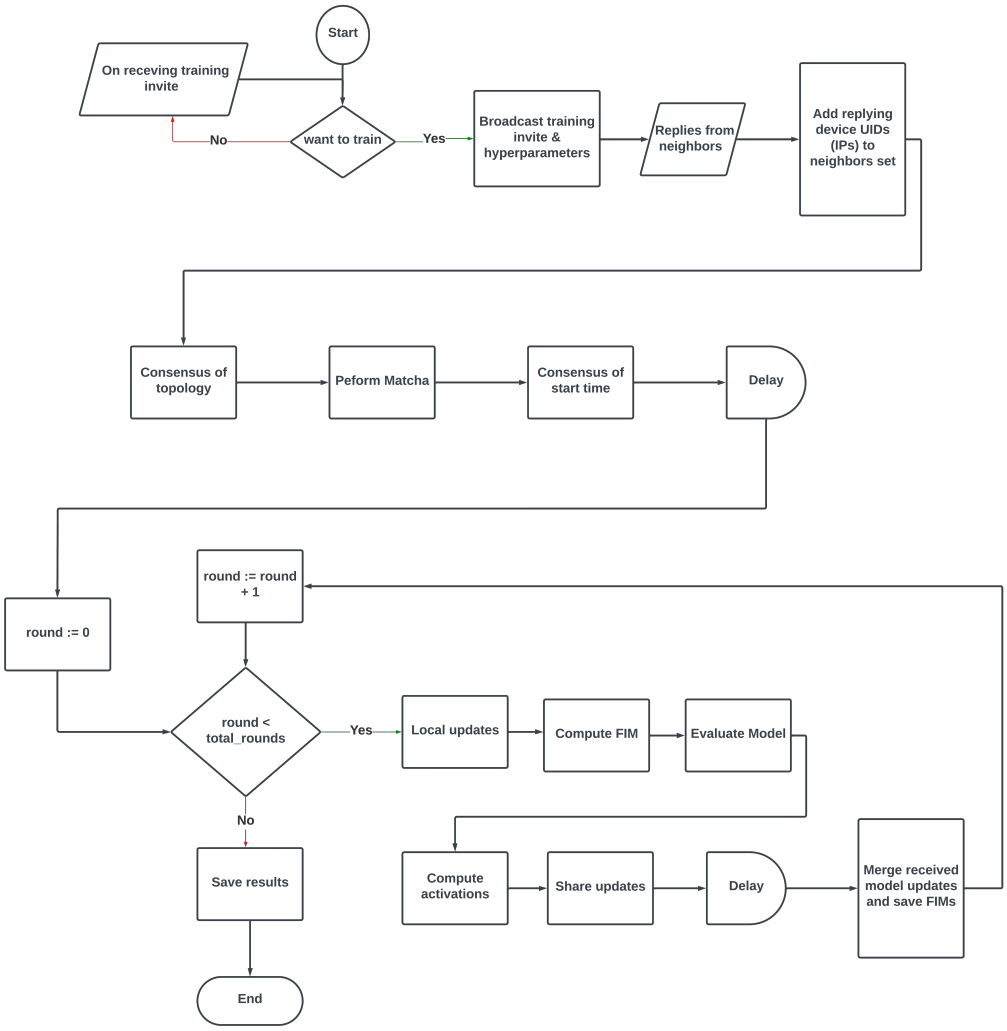


Figure 4.1: The image illustrates the control flow within our framework. In our proposed framework, any edge device, starting from its idle state, makes a determination regarding whether to initiate training or not, and then adheres to the control flow depicted above.

4.2 Consensus step

In wireless ad hoc networks, synchronized activities among nodes are crucial. These activities encompass tasks such as clock synchronization, message sequencing, leader node selection, and decision agreements. All of these tasks require nodes to come to a consensus or agreement among themselves. The core of this consensus challenge involves each node putting forth a proposed value, followed by a process that leads to a collective agreement on a shared result. Although this concept might sound simple in theory, its practical implementation is quite complex [48].
Synchronous distributed algorithms work in rounds, where nodes exchange messages concurrently:

Algorithm 2: General Structure of a Distributed Algorithm

```
/* Each node (v) performs these actions concurrently:          */
1 Send messages to neighbors.
2 Receive messages from neighbors.
3 Perform local computations for the next round.
4 Repeat until termination.
```

Our specific needs involve reaching agreement on two aspects: defining the network's structure and establishing a common start time for training. Remarkably, both tasks can be reduced to identifying the highest value (extrema) within the network. For time agreement, we use the maximum finding function as the local computation in Algorithm 2, and for network topology, we use the graph composition function, merging all the received graphs.
To facilitate this, we defer to simple gossip protocols, but leader election algorithms like the bully algorithm, which work by finding maximum values in the network, are also suited for this task [49]. More complex algorithms like Tora can also be adapted for achieving consensus on maximum values, offering flexibility in implementation based on specific requirements [50], [51].

4.3 Matcha Step

The Matcha algorithm operates after establishing consensus on network topology. Once the graph structure is fixed, new participants are no longer accepted, and devices stop adding new neighbors. The algorithm begins by initializing an empty set, denoted as M, to store encountered matchings. Next, the algorithm enters an iterative loop as long as edges remain in set E. In each iteration, it computes a maximal matching, m, using the $maximal\_matching(E)$ subroutine. The specifics of this subroutine are not detailed here, but it can follow graph coloring principles. The resulting matching, m, is added to the collection M, and the associated edges are removed from E. As the algorithm progresses, it builds a temporary set, X, by creating triples for each matching $m \in M$: $(m, laplacian(m), rand())$. Here, $laplacian(m)$ refers to the Laplacian matrix for matching m, and $rand()$ generates a random value in [0, 1], adding diversity to initial probabilities.

The algorithm then enters the optimization phase, aiming to determine optimized probabilities, P. It employs constrained optimization, compatible with various local solvers. The primary goal is to maximize the second-smallest eigenvalue (λ_2) from the sum of products between p (from triples in X) and the corresponding Laplacian matrix, l. This optimization prioritizes matchings contributing to a higher λ_2. Constraints include ensuring the sum of p values doesn't exceed a set capacity limit, $C_b|M|$, and each p value remains in [0, 1]. When C_b is 1, all probabilities are set to 1, but with a fraction of the communication budget, probabilities reflect network matching importance.

Algorithm 3: Matcha Algorithm

input : Connected Graph $G(V, E)$
output: P

```
M := ∅
while |E| > 0 do
    m := maximal_matching(E)
    M := M ∪ {m}
    E := E \ m
X := {(m, laplacian(m), rand()) | m ∈ M}
Solve for P using equaiton (2.3)
P := {(m, p) | (m, l, p) ∈ X}
```

Furthermore, the Activations Algorithm in Matcha aims to select specific vertices in a graph for activation, determining neighbors for sharing model updates. It focuses on choosing a subset of vertices, denoted as $\bar{V} \subset V$, based on associated matching probabilities.

Algorithm 4: Activations Algorithm

input : v vertex representing device, Probabilities P
output: $\bar{V} \subset V$

```
V̄ := ∅
foreach (m, p) ∈ P do
    if p < rand() then
        foreach (v1, v2) ∈ m do
            if v1 = v then
                V̄ := V̄ ∪ {v2}
            if v2 = v then
                V̄ := V̄ ∪ {v1}
```

The algorithm starts with an empty set, $\bar{V}$, meant to contain the activated vertices. For each probability p corresponding to a matching m in the set P, the algorithm makes a decision by generating a random value and comparing it to p. If the random value is smaller than p, the algorithm examines matching m. Within this matching,

it looks at each edge (v_1, v_2) it comprises. If vertex v_1 corresponds to the input device's vertex v, the algorithm adds vertex v_2 to the $\bar{V}$ subset. Similarly, if vertex v_2 corresponds to device vertex v, it includes vertex v_1 in $\bar{V}$.
In summary, the algorithm builds the $\bar{V}$ subset, indicating suitable vertices for sharing model updates based on assigned probabilities. It's important to ensure that each device coordinates a random seed, as a training hyperparameter or using the Consensus step mentioned earlier.

4.4 Local Updates

The Local Update Algorithm enhances model parameters by incorporating local data and information from neighboring devices, iteratively refining the model. Each iteration, focuses on a minibatch from the local dataset, aiming to minimize the objective function $\hat{F}(w)$:

$$\min_{w} \hat{F}(w) = F(w) + \lambda \sum_{(\bar{w},\bar{I})\in N} (w - \bar{w})^T \mathrm{diag}(\bar{I})(w - \bar{w}) \tag{4.1}$$

Here, $F(w)$ is the base loss function on the local dataset, and the regularization term integrates insights from neighbors through pairs $(\bar{w}, \bar{I})$ from set N. $\mathrm{diag}(\bar{I})$ represents a diagonal matrix derived from the received FIM $\bar{I}$. The algorithm's objective is to find w that minimizes $\hat{F}(w)$, stopping if a time limit is reached for each minibatch update.

Algorithm 5: Local update Algorithm

input : Parameter and FIM pairs from neighbors N
output: Parameters w

```
Start the timer.
foreach minibatch B ∈ local dataset D do
    if timeout then
        Stop.
    Find w using equation (4.1)
```

And, Algorithm 6 computes the Fisher information matrix. It uses a timer to track computation time and starts with an empty matrix I filled with zeros.
In the main loop, it iterates through training data samples X within a set time limit. For each iteration, it randomly selects a sample x, calculates logits with the softmax function, and computes gradients θ by taking derivatives of the logits with respect to model parameters. The algorithm updates the Fisher Information Matrix by adding the element-wise squares of computed gradients for each sample, capturing information about how sensitive the model is to input variations. After processing all samples, the algorithm normalizes the Fisher Information Matrix by dividing it by the total number of samples n to ensure an average sensitivity across the dataset. Finally, it returns the computed Fisher Information Matrix, which estimates model parameter uncertainty and guides the optimization process.

Algorithm 6: Compute Fisher Information Matrix

input : Neural network model *model*, training data X, number of samples n
output: Fisher information matrix I

```
1  Start the timer. ;                          /* Start timing the computation */
2  Initialize an FIM I as a list of zeros.
3  foreach i ∈ [1, n] do
4      if timeout then
5          Stop. ;                                        /* Stop if timeout */
6      Randomly select sample x from X.
7      Compute logits using log(softmax(model(x))).
8      Compute gradients θ: θ ← ∇_model(logits).
9      Update I by adding the element-wise square of θ.
10 Normalize I by dividing by number of samples n.
11 return I ;          /* Return the computed Fisher Information Matrix */
```

This description provides an adequate understanding of the algorithm structure. For more detailed specifics, refer to the "Implementation" section, which explores system components and interactions in greater depth. Design decisions, such as enhancing resilience against device dropouts or implementing more robust consensus mechanisms, are discussed in the concluding section.

Chapter 5
Implementation

This chapter offers an in-depth exploration of the steps taken to implement the MatchCurv framework. We will examine the fundamental components of the framework and their role in enabling decentralized federated learning.

5.1 Implementation for Raspberry PIs

The development of the MatchCurv framework was a meticulous process, driven by the need to ensure its seamless operation across various scenarios, including both simulation environments and Raspberry Pi devices. We chose Python 3 as our programming language due to its user-friendly features and widespread usage, making it an accessible choice for developers and researchers alike.
One of the key design considerations was optimizing resource utilization. To achieve this, we implemented a multi-threading approach, which proved to be highly efficient. By using multiple threads, we could effectively distribute communication and computation tasks. Within this framework, distinct threads were responsible for managing incoming and outgoing packets. These threads coordinated their efforts through the use of a packet queue, allowing for the storage and retrieval of data as needed, all while adhering to essential thread safety protocols.

5.1.1 Ensuring Consistency Across Devices

Consistency across all devices was a paramount concern. To achieve this, we mandated that all devices initialize with the same random seed at the outset of the algorithm. This synchronization was crucial as it guaranteed that all devices computed identical activations at the end of each training round. This consistency prevented variations in activations and ensured the correct updating of neighboring devices during training rounds. The synchronized random seed approach was particularly well-suited for Raspberry Pi devices and eliminated the need for complex coordination mechanisms during the training process.
In addition to synchronized random seeds, we also addressed other coordination requirements among our devices. In our experimental setup, we facilitated communication among Raspberry Pis through the use of a standard Wi-Fi router, creating a fully connected training network where each device could communicate with every other device. Although an on-the-fly ad hoc network might be an ideal choice in some scenarios, we opted for the simpler approach of using a router for the sake of implementation ease. It's worth noting that, in this specific implementation, we assumed there were no dropouts during training. However, we acknowledged the possibility of exploring the framework's functionality under dropout conditions in future work.

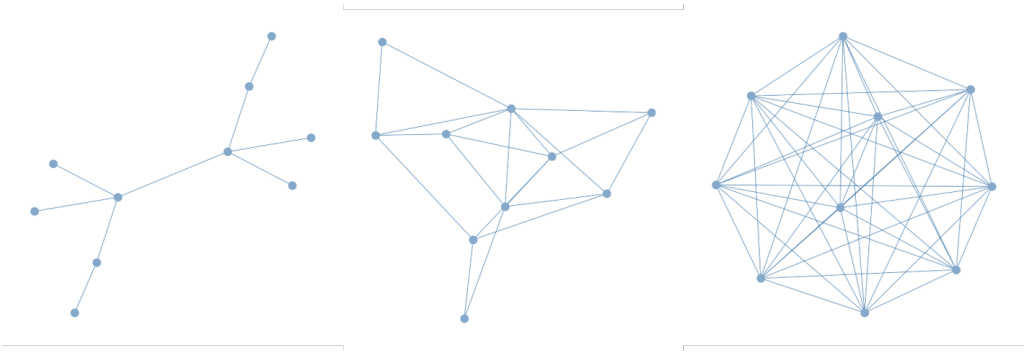


Figure 5.1: This figure illustrates various network topologies employed in assessing our framework. Our implementation provides the flexibility for all devices to establish direct communication with each other, forming a fully connected network. However, during training, devices adhere to specific topologies rather than maintaining constant full connectivity. Beginning from the left, we have a sparsely connected topology with the minimum required edges to maintain graph connectivity. As we progress to the right, the number of edges in the graph increases, culminating in a fully connected graph on the far right.

Furthermore, we took steps to specify preset graph topologies and model weights as initial values for the neural networks trained using our framework. While not mandatory, this approach ensured reproducible results when using these presets. Importantly, all relevant information, including critical hyperparameters such as the communication budget and data distribution, was explicitly defined in a configuration file for the training process. This configurability allowed for adaptability and customization of the framework. If presets were not specified in the configuration file, the processes would initialize model weights randomly and dynamically create a topology based on the replies received from neighboring devices, provided they could establish communication without relying on the router.

5.1.2 Future proofing with Modularity

Furthermore, to underscore the framework's adaptability and future-proofing, our implementation was intentionally designed to be modular and independent of specific Python libraries. While Figure 5.4 outlines the chosen Python libraries for our implementation, it's important to note that this framework offers flexibility and can readily accommodate alternative library choices. Modifying the framework, whether to enhance communication capabilities, eliminate the need for a router, or switch to a preferred machine learning library, primarily involves adjustments to specific modules, such as the communication or model module. Importantly, these modifications do not compromise the core functionality of the framework, ensuring its versatility and resilience in diverse decentralized federated learning scenarios.

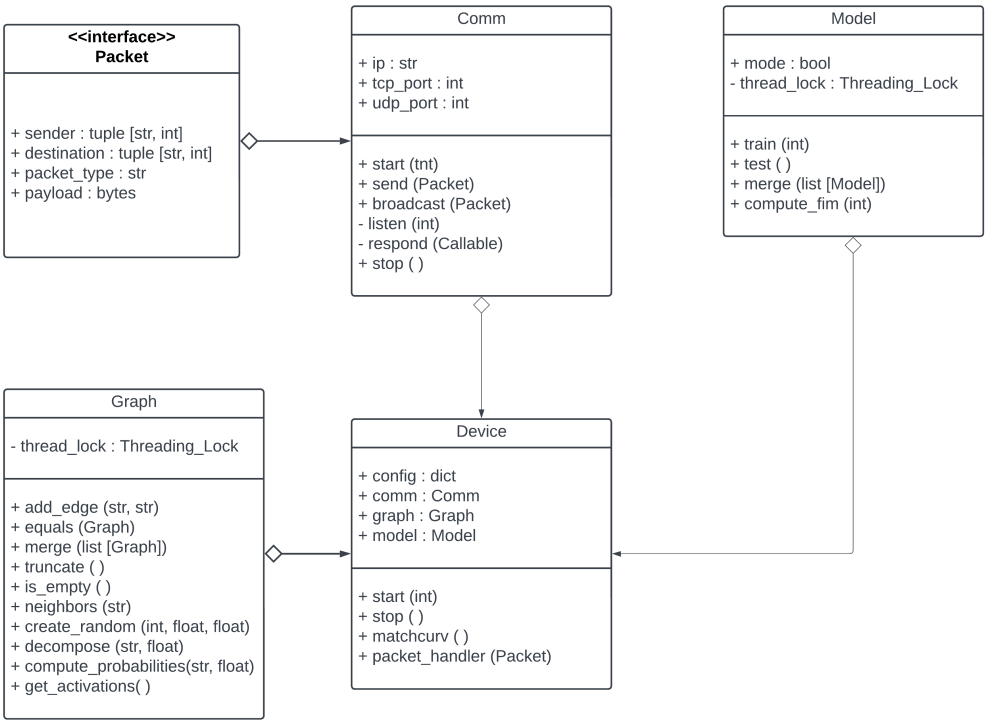


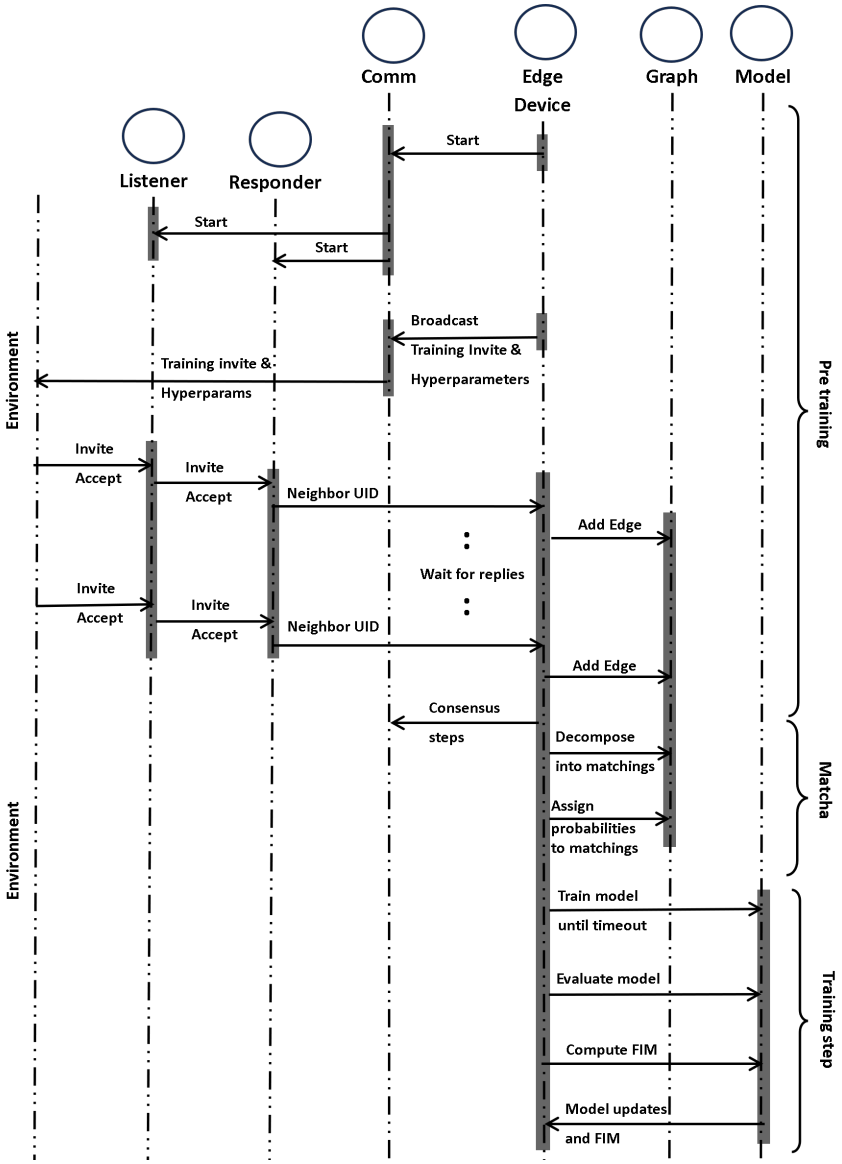
Figure 5.2: The figure provided illustrates the module dependencies within the MatchCurv system, offering insights into the core components of the framework. Each module plays a distinct role in ensuring the functionality and performance of the system. The Comm Module is responsible for handling all aspects of communication with other devices. It manages data exchange, message passing, and updates between devices in the network. The Model Module takes charge of critical tasks related to machine learning. This includes model training, evaluation, and the computation of the Fisher information matrix. It serves as the core module for machine learning operations within the framework. The Graph Module is dedicated to network-related functions. It handles tasks such as creating the network graph structure, merging graphs from neighboring devices, decomposing the graph into matchings, and executing matching operations. The Edge Device Class serves as a central interface connecting all other modules. It not only initiates processes within these modules but also ensures their proper termination at the conclusion of the algorithm's execution. This class acts as the coordinator of decentralized federated learning, orchestrating various activities within the framework. The Edge Device module plays a pivotal role in managing processes and activities within the algorithm, effectively overseeing decentralized federated learning. It integrates communication, model training, and network operations, facilitating the seamless execution of the MatchCurv framework.

5.1.3 Creating heterogeneity among Devices

To streamline the evaluation process and gain a comprehensive understanding of the framework's capabilities, we took into account both statistical and system heterogeneity among the devices. Simulating labels for non-IID environments involved the strategic distribution of data samples to different devices. On the other hand, simulating system heterogeneity required a different approach. We artificially introduced delays for specific devices by making them "sleep" for a portion of the time it takes for devices to complete a specific number of training epochs. This approach effectively replicated varying processing speeds and capabilities among different devices. When selecting these "stragglers" for introducing delays, we did so randomly, with the only constraint being that if devices were to be dropped from training, the resulting network must remain connected.
Additionally, ensuring that model performance was evaluated on previously unseen samples necessitated having samples from all labels in the dataset. To meet this requirement, we allocated a specific portion of the dataset exclusively for testing purposes. These samples were reserved solely for evaluating the model's performance and were not involved in the training process. Importantly, these implementation details were not only relevant for our physical devices, such as Raspberry Pis, but also seamlessly applicable to the simulation environment. This consistency ensured that our evaluation process remained robust and reliable across different platforms.

5.2 Implementation for Simulation

Evaluating the framework's performance with a large number of physical devices alone would be impractical due to logistical constraints. To overcome this limitation, our framework has been meticulously designed to operate seamlessly on both physical devices and within a simulation environment. In the simulation environment, subprocesses simulate the behavior of edge devices, allowing for scalability and flexibility in testing. This adaptability is achieved with minimal code adjustments, such as specifying local addresses for communication rather than physical device addresses. It ensures that the same framework can be used for experiments in both physical and simulated environments, simplifying testing and evaluation. However, when selecting the model to train, it's crucial to consider the capabilities of the device, such as the available memory and network quality. The suitability of the model for a particular device should align with its hardware limitations to ensure efficient execution. In our work, the experiments were conducted on a server that demonstrated the capacity to handle over 100 subprocesses emulating the behavior of edge devices. This setup allowed us to evaluate the framework's performance under realistic and scalable conditions.
Within a single physical device, the inherent capability of processes to communicate with all other processes enables the creation of a fully connected network topology among the training devices. This approach effectively replicates the communication challenge encountered in the Raspberry Pi implementation, where reliance on a router was essential. Consequently, the solution previously deployed for Raspberry Pis, which enabled efficient communication, can be seamlessly applied in this


Comm
Edge Device
Graph
Model
Listener
Responder
Start
Start
Start
Broadcast
Training Invite &
Hyperparameters
Environment
Training invite &
Hyperparams
Invite
Accept
Invite
Accept
Neighbor UID
Add Edge
Wait for replies
Invite
Accept
Invite
Accept
Neighbor UID
Add Edge
Consensus
steps
Decompose
into matchings
Assign
probabilities
to matchings
Environment
Train model
until timeout
Evaluate model
Compute FIM
Model updates
and FIM
Pre training
Matcha
Training step

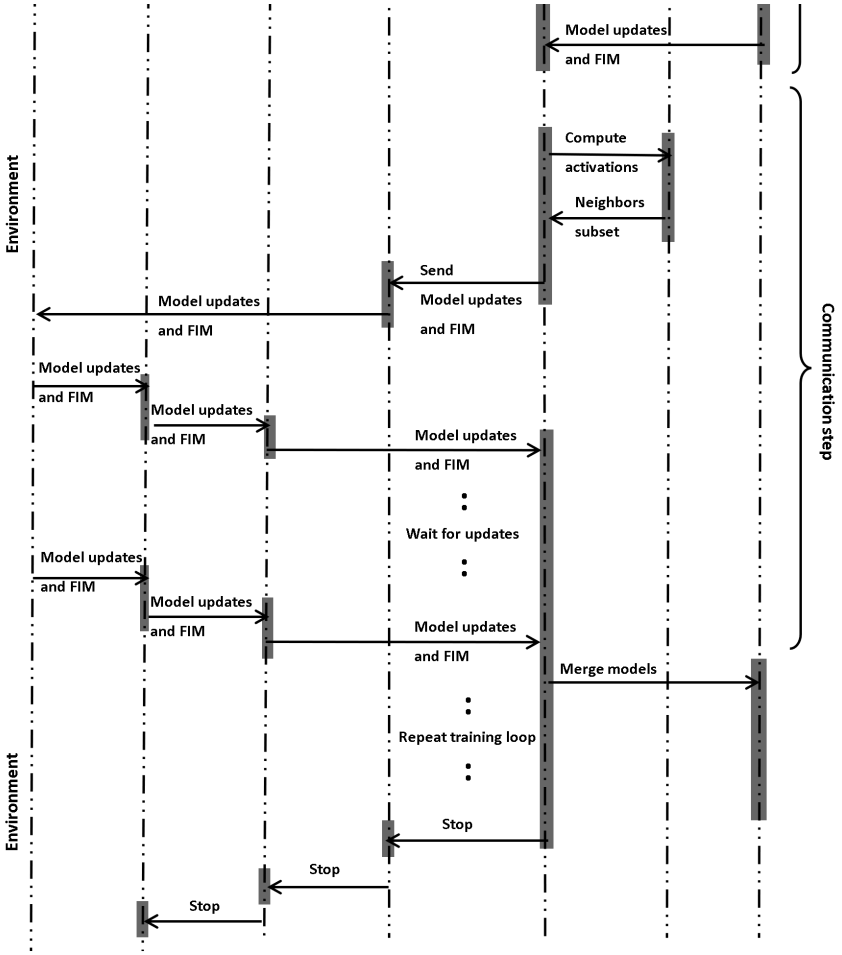


Figure 5.3: The provided diagram visually represents the dynamic connections between various processes that come into existence during the execution of the MatchCurv algorithm. At the core of this orchestration is the Edge Device instance, which not only initiates these processes but also plays a pivotal role in synchronizing their actions throughout the training process. This orchestration spans the entirety of the training procedure, guaranteeing smooth coordination and communication among the diverse subprocesses involved.

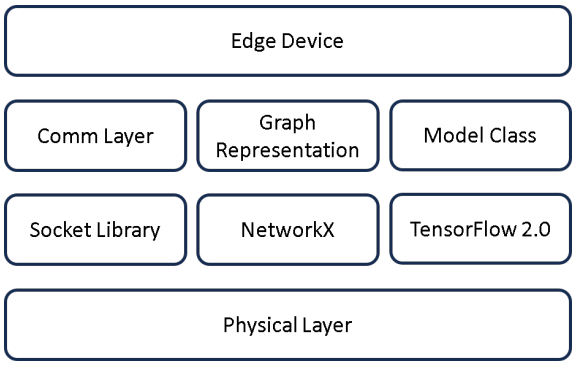


Figure 5.4: This figure illustrates the hierarchical structure of the MatchCurv system, highlighting its various layers and essential components. The deepest layer is the physical layer, which uses standard Python libraries to support the entire system. Above that is the Communication Module, crucial for interactions between devices and processes. It uses a socket-based architecture, supporting various communication scenarios like TCP device-to-device and UDP broadcasting. Next to the Communication Module is the Graph Module, which represents interconnected devices within MatchCurv. The Model Module supports synchronous and asynchronous training modes, fostering collaborative learning among system components. At the topmost layer is the Edge Device Module, which emulates device behavior and oversees interactions, communication, and collaborative learning. These layers and modules form the foundation of MatchCurv's functionality and system architecture.

context. To simulate a random network topology, we have established predefined graph presets. Processes adhere to these presets, following the specified connections while disregarding those not outlined in the presets. The creation of these random topologies is facilitated using networkx's implementation of connected Watts-Strogatz graph generation.
Furthermore, as previously mentioned, our implementation relies on all devices using the same random seed to ensure consistent activations during each training round. It's not just the random seed that requires synchronization; other random events, such as the selection of stragglers, need to be synchronized within the simulation environment so that every process or edge device knows if it should behave as a straggler. However, setting a random seed directly within a simulation presents challenges because subprocesses share the same pseudo-random number generator. To address this challenge, we employ a random preset, which consists of a predetermined list of random numbers. Consequently, we conduct experiments with different random, graph, and model presets both within the simulation and on the Raspberry Pis to maintain consistency. This approach allows us to validate the results obtained within the simulation environment using physical Raspberry Pi devices, ensuring the robustness and reliability of our framework.

That summarizes the core details of our framework's implementation, with further intricacies about the evaluation to follow.

Chapter 6
Evaluation

In this chapter, we delve into the empirical evidence that highlights the performance advantages of utilizing the MatchCurv framework over Periodic Decentralized Stochastic Gradient Descent (PD-SGD). Our exploration begins by elucidating the strengths of each of these solutions in addressing specific challenges. Subsequently, we conduct a comprehensive analysis of the overall performance of our framework.
To establish a baseline for our experiments, we trained a multi-class logistic regression (MCLR) model using the MNIST dataset, running it for a total of 1000 epochs using conventional ML. The results of these baseline experiments serve as a reference point, with the model achieving an evaluation/test accuracy of 92%. This accuracy figure sets the standard against which we will compare and assess the results of our subsequent experiments.
Furthermore, we expanded our baseline by training a classification model on the fashion-MNIST dataset. This time, we employed a multi-layer perceptron (MLP) architecture and trained it for 1000 epochs. The outcome of this extended baseline experiment yielded an accuracy rate of 84%. Both of these baseline values serve as essential benchmarks, enabling us to gauge the improvements and advantages conferred by the MatchCurv framework in comparison to established methodologies.

6.1 Performance under Statistical Heterogeneity

FedCurv was originally introduced as a solution to address statistical heterogeneity in federated learning. However, our research focuses on decentralized federated learning and introduces surface-level but potentially impactful modifications to the original work. These modifications leave the framework's performance in this decentralized environment uncertain.
To assess the effectiveness of these modified equations, we conducted experiments. Specifically, we trained a multi-class logistic regression (MCLR) model for recognizing handwritten digits using the MNIST dataset. The dataset is distributed in three different ways across 10 devices:

- IID (Independent and Identically Distributed): In this scenario, each device receives 10% of the samples from all digits 0 to 9.
- Non-IID - 5: Here, each device is allocated 20% of the samples from 5 specific digits Importantly, the samples across all devices still correspond to the initial dataset. The distribution of samples is non-intersecting.
- Non-IID - 2: In this case, each device is assigned 50% of the samples from 2 specific digits. Similar to the Non-IID - 5 setting, the samples across all devices comprise the initial dataset.

The devices were interconnected through 20 edges, as depicted in Figure 5.1. To isolate the impact of non-IID data distribution and assess the potential improvement

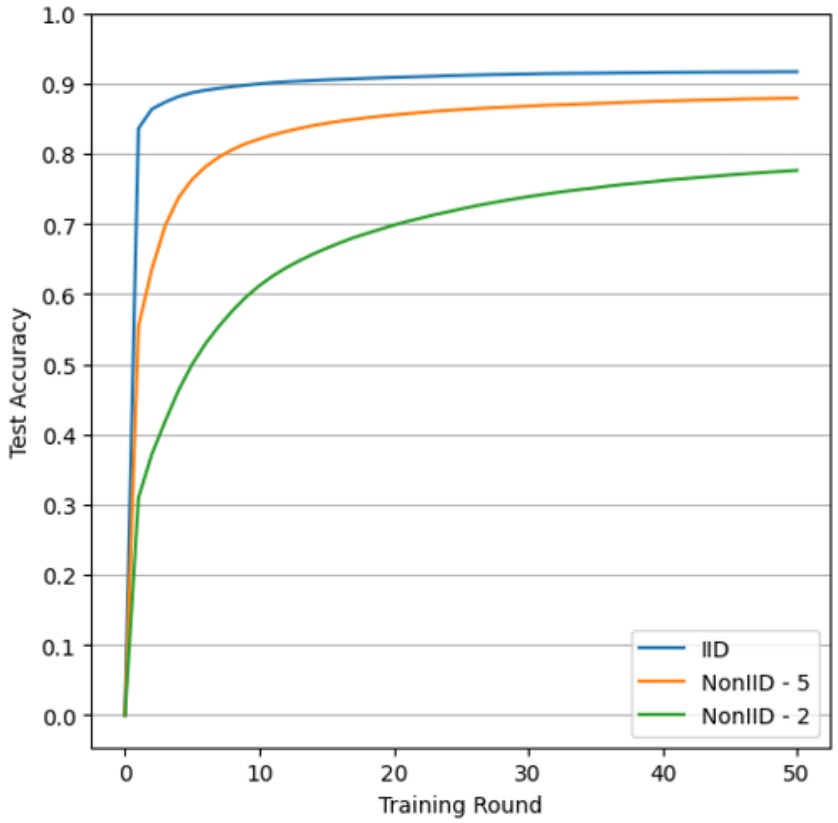


Figure 6.1: This plot illustrates the model's performance under three different settings: IID, NonIID5, and NonIID2. In the NonIID5 scenario, the performance nearly matches that of the IID setting, whereas in the NonIID2 scenario, the performance falls significantly short.

in model accuracy achieved by applying the penalty from equation 4.1 compared to plain stochastic gradient descent (SGD), we assumed there were no stragglers and allocated a 100% communication budget (further details are available in the Appendices).
When the devices underwent 50 rounds of DFL training, the performance of the models significantly decreased in both non-IID environments compared to the IID setting. The IID setting achieved a classification accuracy of 91% on previously unseen 10,000 samples, closely matching the 92% performance reached by a model trained on a single device using conventional machine learning with the same MCLR model and initial weights. However, when the environment transitioned to Non-IID-5, the performance dropped to 87%, and further to 77% in the Non-IID-2 scenario. This substantial performance drop highlights the challenges posed by non-IID settings compared to IID.

6.1.1 Effect of Periodic Communication

This decline in performance is further exacerbated by certain hyperparameters introduced in DFL. For instance, we initially trained for 10 epochs in each training

Config	Test Accuracy (%)
IID	91.69
NonIID - 5	87.93
NonIID - 2	77.62

Table 6.1: The table below illustrates the decline in test accuracy, specifically in classifying samples not used during training, as the data distribution deviates from being identically and independently distributed (IID). It's worth noting that the greater the dissimilarity in label distribution among devices, the more significant the drop in accuracy becomes. In the NonIID - 2 setting, the accuracy reaches only 77%, falling considerably short of the 92% performance achieved by a conventional ML model.

round before sharing the model weights. Subsequent experiments aimed to explore how the number of epochs in the training round affects model performance in the presence of non-IID data, particularly in the non-IID-2 scenario.
With just 1 epoch in each training round, we improved the accuracy from 77% (with 10 epochs) to 89%. Conversely, increasing the number of epochs to 50 led to a sharp drop in accuracy to 59%. This illustrates the critical importance of selecting appropriate hyperparameters in DFL, especially in the presence of challenges like non-IID data. Other factors like the training topology, which may not always be under our control, also play a significant role. A denser and more robust network of training devices is essential to ensure uninterrupted training. We further explore the effects of device topology on model performance.

No. of Epochs	Test Accuracy (%)
1	89.98
10	77.62
50	59.60

Table 6.2: This table presents the impact of the number of training epochs on testing accuracy. Our hypothesis suggests that as the number of training epochs in a round increases, there is a greater likelihood of the model overfitting to local data and consequently forgetting the global parameters learned in previous training rounds. The results confirm this hypothesis, particularly in the context of non-IID data, where devices rely on insights from other devices due to the distributed nature of the data.

6.1.2 Effect of Topology

Figure 5.1 illustrates the three topologies used in our experiments. The sparsely connected topology resulted in a mere 51% evaluation accuracy, a stark drop from the baseline of 77% in the non-IID-2 scenario with a denser graph. However, with a fully connected topology, we observed an improvement in accuracy to 88%, getting closer to the accuracy achieved with conventional machine learning. It's worth noting that the fully connected topology resembles centralized federated learning

(CFL), where devices can be thought of as being connected to each other by proxy of the central entity.

Topology	Test Accuracy (%)
Full	88.82
Dense	77.62
Sparse	51.85

Table 6.3: This table demonstrates a clear trend: as the training topology becomes sparser, the model's testing accuracy progressively decreases.

6.1.3 Effect of Neighbor Quality

While the previous section focused on how network density affects average model performance, the quality of the devices from which we receive updates can also impact model performance. In non-IID settings, we rely on insights from other devices to handle samples from all classes/labels in the dataset effectively. As our immediate connections provide this knowledge at every turn in a training round, it becomes interesting to examine how the data distributions in the neighbors affect overall model performance. To explore this, we created two different graph presets used for training. One represented a topology where devices sought connections with others having different data distributions, while the other sought connections with devices having identical data distributions. With these presets, we observed the model's performance using the same experiments as before, witnessing a performance drop from 83% to 59% when transitioning from good neighbor quality to poor neighbor quality.

Neighbors	Test Accuracy (%)
Diverse	83.87
Identical	59.14

Table 6.4: This table showcases how a reduction in the diversity of data distributions among neighboring devices is correlated with a decline in model performance.

6.1.4 Effect of MatchCurv

In the "Related Work" chapter, we briefly mentioned some existing solutions for statistical heterogeneity, such as performance-based neighbor selection or optimizing the local update frequency. These solutions, however, overlook the underlying motif of the aforementioned problems, which is catastrophic forgetting. In scenarios with non-IID devices, the challenge lies in devices relying on insights from other devices to enhance accuracy on labels or classes that are underrepresented in their local dataset. When dealing with poor neighbor quality, where neighboring devices have identical datasets, we depend on insights from devices farther away in the network. However, these insights are not simply relayed; rather, at each training round, model

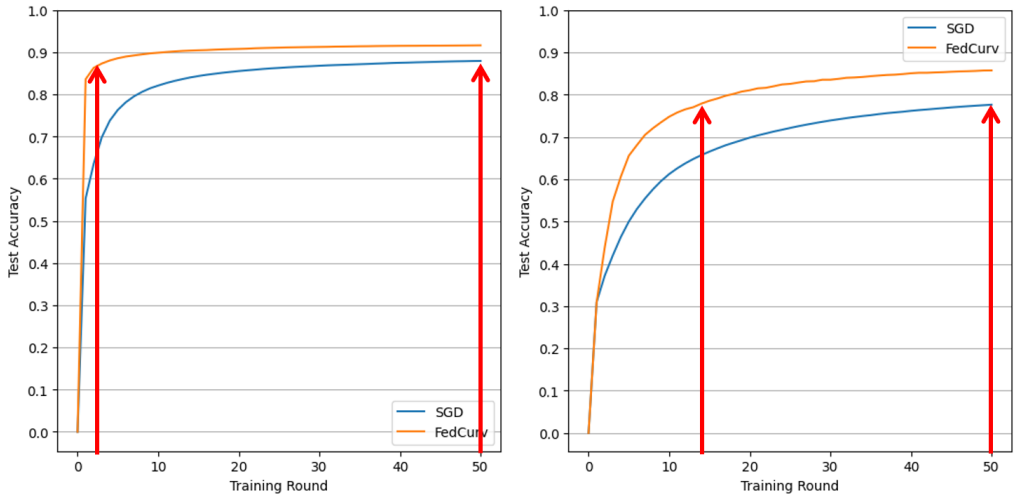


Figure 6.2: On the left side, you'll find a plot depicting test accuracy (vertical axis) versus the number of training rounds (horizontal axis) in the NonIID-5 setting. MatchCurv method quickly matches the performance of SGD, as indicated by the red arrows. Notably, it not only achieves this performance rapidly but also attains significantly higher accuracy by the end of the training. On the right side, there's a plot representing the NonIID-2 scenario. The overall trend is similar, but the disparity in test accuracy between MatchCurv and SGD is even more pronounced in this case.

updates are integrated with other updates and further trained upon by devices along the path. This process can potentially dilute the updates before they reach a device that relies on them. Additionally, in sparser graphs, there are fewer paths to reach a device, and having more epochs in a training round further exacerbates the dilution of model updates.
All these challenges contribute to the catastrophic forgetting of model parameters. This brings us to FedCurv. FedCurv has demonstrated its superiority over other existing methods by improving model performance in the presence of non-IID data. However, considering the changes we made to the original work, we aim to showcase the performance of these modifications. We conducted experiments using the same initial settings used to assess the impact of non-IID data, but this time, we applied the penalty from equation 4.1. We search for λ value for FedCurv from $[0, 0.01, 0.1, 1]$ and pick the most performant value.

Data Distribution	Solver	Test Accuracy (%)
NonIID - 5	SGD	87.93
NonIID - 5	MatchCurv	91.64
NonIID - 2	SGD	77.62
NonIID - 2	MatchCurv	85.72

Table 6.5: In this table, we can observe the test accuracy improvements achieved with MatchCurv compared to SGD. Across both Non-IID data settings, there is a noticeable and consistent enhancement in testing accuracy.

With our framework, accuracy improved significantly. In the non-IID-5 setting, the accuracy reached 91%, mirroring the performance in the IID setting. In the Non-IID-2 setting, we gained more than 8 points in accuracy, moving from 77% to 85%. This dramatic improvement isn't just about accuracy; we got to the previously attained accuracy levels much earlier in training. In the case of non-IID-2, we reached the 77% accuracy mark by the 13th round, compared to the 50 rounds it took without FedCurv.

6.2 Performance under Systems Heterogeneity

To assess model performance under systems heterogeneity, we must take into account the variations in computational capabilities among devices. To simulate this behavior, we designate a certain percentage of devices (0%, 25%, or 50%) as stragglers. The selection of which devices become stragglers is random. Stragglers are essentially ignored during the training process because they cannot complete their assigned work in time and share their model updates before others have finished waiting. In essence, stragglers are dropped from training. This exclusion can be problematic, as these devices may possess crucial samples that could contribute to the validation accuracy.

6.2.1 Effect of Stragglers

When dealing with IID data, we typically do not observe a significant decline in performance due to the presence of stragglers. This lack of detriment can be attributed to the fact that IID data ensures samples from all labels are present, helping mitigate the impact of stragglers. However, in scenarios with non-IID data, particularly in the non-IID-2 setting, we notice a decline in performance. For instance, in the presence of 25% stragglers, the accuracy drops from an initial 77% to 60%. This drop becomes even more pronounced, decreasing to 51% accuracy when 50% of the devices are designated as stragglers.

% of Stragglers	Test Accuracy (%)
0	77.62
25	60.11
50	51.65

Table 6.6: The table show cases the effect of stragglers in the network on the test accuracy.

In addressing the challenge of stragglers, we initially employed FedCurv as the modified version we've developed in this work. Using FedCurv alone, we achieved a notable improvement in performance. Specifically, in the presence of 25% stragglers, we reached an accuracy of 68%, and with 50% stragglers, we achieved 66% accuracy. This demonstrates that FedCurv can enhance performance in the presence of stragglers, but we conducted further experiments to explore potential improvements.

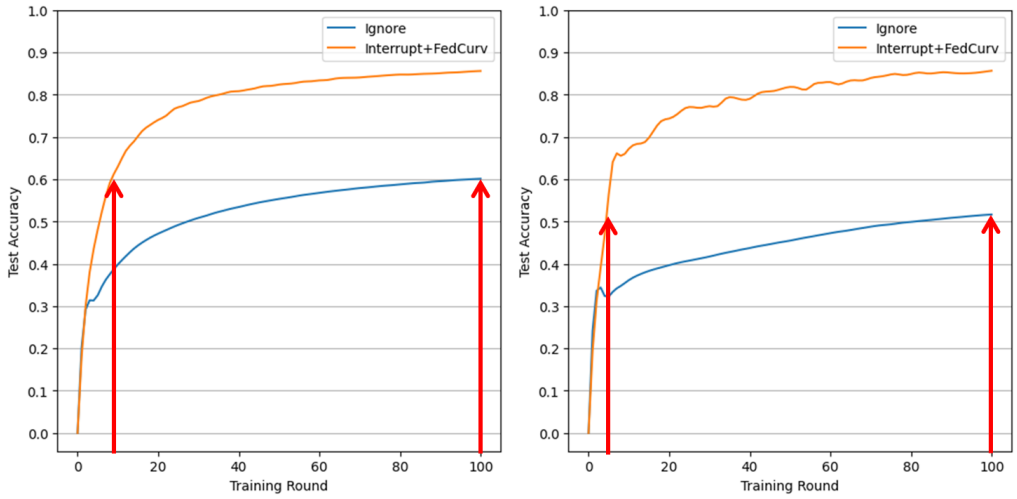


Figure 6.3: By combining the strategies of interrupting stragglers and employing FedCurv, we observe a substantial improvement in performance in both scenarios: one with 25% stragglers (on the left) and another with 50% stragglers (on the right).

6.2.2 Effect of MatchCurv

Our proposed solution involves interrupting devices during training. To clarify, instead of specifying a fixed number of epochs, such as 10 in this experiment, we set a fixed training time. This time is calculated as the average time it takes non-stragglers to complete the training. At the end of this training time, the stragglers interrupt themselves and submit whatever partial work they have completed within this timeframe.

% of Stragglers	Configuration	Test Accuracy (%)
25	Ignore	60.11
25	FedCurv	68.39
25	Interrupt	77.97
25	Interrupt + FedCurv	85.59
50	Ignore	51.65
50	FedCurv	66.83
50	Interrupt	78.62
50	Interrupt + FedCurv	85.64

Table 6.7: This table clearly illustrates the performance benefits of both interrupting stragglers and implementing FedCurv, both when used individually and when combined. Importantly, these two solutions not only do not conflict with each other but also complement each other, leading to improved results.

This approach led to a significant performance increase. In the presence of 25% stragglers, we improved the model's accuracy from 60% to 77%. When dealing with 50% stragglers, the accuracy reached an impressive 78%, even surpassing the baseline performance of 77% accuracy. However, the most promising results emerged when combining both solutions. By accepting partial work from stragglers and utilizing

FedCurv, we achieved even better performance, with an accuracy of 85% in both scenarios.

6.3 Performance under a Communication budget

To alleviate the communication burden on devices in Decentralized Federated Learning, we introduce the concept of a communication budget, inspired by Matcha. In this approach, the training topology is divided into predefined matchings, and these matchings are assigned activation probabilities based on the communication budget (Cb). When the Cb is set to 100%, all matchings in the graph are activated (selected for parameter sharing) during each training round. If the Cb is reduced, let's say to 10%, the activation probabilities are adjusted accordingly, resulting in a communication load close to 10% of that with a 100% Cb. With these probabilities assigned randomly, we conducted several experiments in a non-IID-2 setting.

6.3.1 Effect of Random neighbor selection

We conducted three experiments with different Cb values: 1 (100% Cb), 0.5 (50% Cb), and 0.25 (25% Cb). When the communication budget was reduced by 50%, we observed a 21% drop in accuracy, decreasing from 77% to 56%. Further, reducing the communication budget to 25% led to an even lower accuracy of 43%. However, by incorporating FedCurv, we managed to improve these accuracies to 71% and 67%, respectively. While FedCurv significantly enhances performance in this scenario, there may still be room for further improvement.

Communication Budget	Test Accuracy (%)
100%	77.62
50%	56.31
25%	43.98

Table 6.8: The introduction of a communication budget results in a significant drop in performance when the budget is insufficient. This table effectively showcases this effect.

6.3.2 Effect of MatchCurv

The Matcha framework employs the spectral graph property, specifically algebraic connectivity, to strategically assign probabilities to the matchings. In our implementation, we incorporate Matcha and follow suit. By using algebraic connectivity to determine these probabilities, we achieved performance levels of 75% accuracy with a 50% communication budget and 69% accuracy with a 25% Cb. These results outperformed the performance gains obtained solely from using FedCurv.
However, as we observed earlier, combining solutions often leads to even greater improvements in performance. When we combined Matcha with FedCurv, we achieved an accuracy of 84% with a 50% Cb, surpassing the baseline performance, and 75%

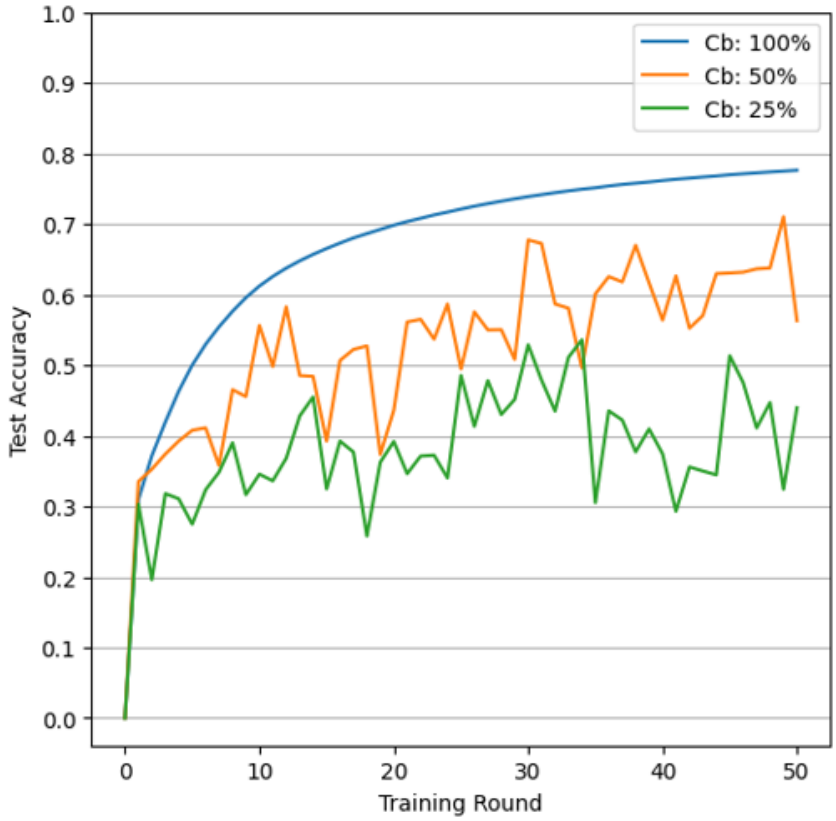


Figure 6.4: The plot illustrates how the communication budget impacts the model's performance. Additionally, it's evident that the curves representing scenarios with less than 100% communication budget exhibit greater fluctuations and irregularities, marked by numerous peaks and valleys. These fluctuations are a result of limited or reduced communication during those rounds.

Communication Budget	Configuration	Test Accuracy (%)
50%	Random	56.32
50%	FedCurv	71.81
50%	Matcha	75.48
50%	Matcha + FedCurv	84.22
25%	Random	43.98
25%	FedCurv	67.53
25%	Matcha	69.83
25%	Matcha + FedCurv	75.43

Table 6.9: This table highlights the performance improvements achieved by the Matcha method and FedCurv, even when communication is reduced. Individually, they mitigate the performance loss caused by reduced communication to a certain extent. However, when combined, their synergistic effect leads to even greater performance improvements.

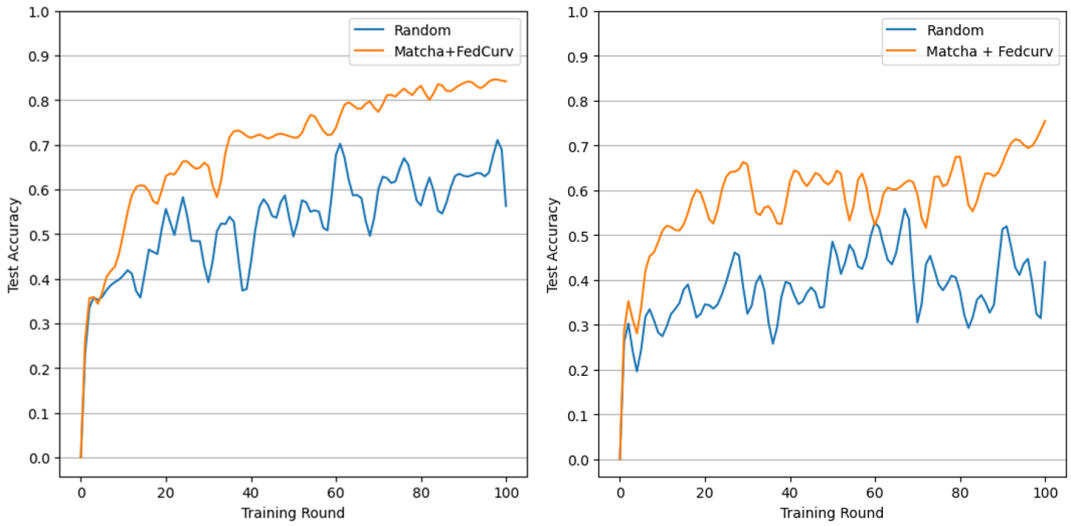


Figure 6.5: In both plots on the left (50% communication budget) and right (25% communication budget), we observe the impact of MatchCurv on the model's performance. Notably, there's a significant difference in the smoothness of performance fluctuations between MatchCurv and SGD, with MatchCurv exhibiting much smoother variations.

accuracy with a 25% Cb, coming close to the baseline accuracy. This combination of Matcha and FedCurv proved to be highly effective in enhancing model performance in DFL with reduced communication budgets.

6.4 MatchCurv vs PD-SGD

So far, we have seen the effect on model performance in isolated scenarios. To comprehensively evaluate our framework's resilience under real-world scenarios, we introduce both statistical and systems heterogeneity into the decentralized federated learning environment while also imposing a communication budget. These conditions represent ideal grounds for assessing our framework's robustness. We conduct experiments in two scenarios:

- Moderate Scenario: In this scenario, we select moderate hyperparameters based on our previous observations. We use a denser topology with 10 epochs (equivalent time) in each training round. Among the devices, 25% are designated as stragglers, and we allocate a 50% communication budget. The data distribution follows a non-IID-5 pattern, and the experiment runs for a total of 100 rounds.
- Extreme Scenario: In the extreme scenario, we set hyperparameters to their most extreme values, as observed earlier. The topology is sparse, and each training round consists of 50 epochs. Half of the devices (50%) are designated as stragglers, and the communication budget (Cb) is limited to 25%. The data distribution is non-IID-2, and the experiment runs for only 20 rounds.

In the moderate environment, PD-SGD retains an accuracy of 77%, while when utilizing straggler interruption, Matcha, and FedCurv individually, we achieve 83%,

Scenario	Data Distr.	Epochs	Stragglers	Comm. Budget	Topology
Moderate	NonIID - 5	10	25%	50%	Dense
Extreme	NonIID - 2	50	50%	25%	Sparse

Table 6.10: This table describes the both scenarios mentioned above.

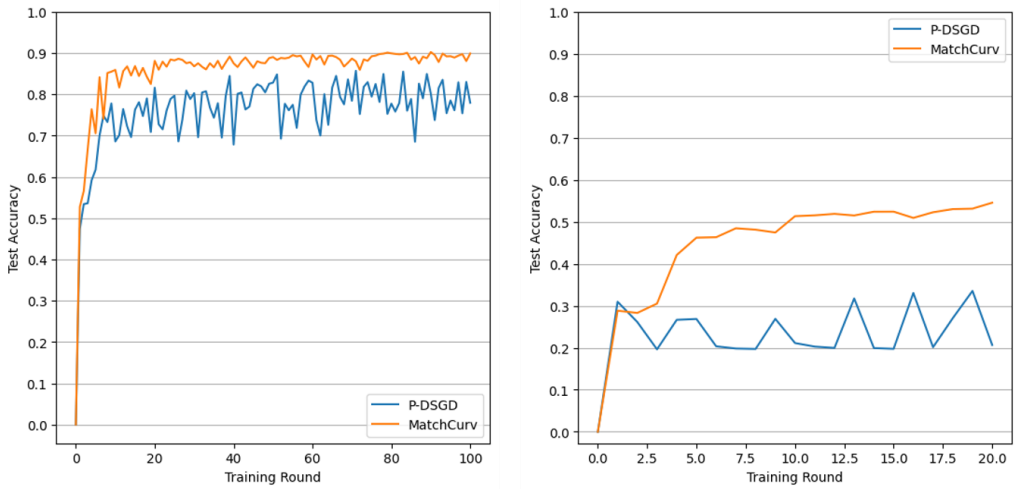


Figure 6.6: The plots above demonstrate how MatchCurv enhances the performance of the MCLR model in both moderate (on the left) and extreme (on the right) scenarios.

85%, and 86%, respectively. However, by combining all these solutions, we optimize the model's performance. With MatchCurv, we achieve an accuracy of 89%, which comes even closer to the baseline machine learning (ML) performance of 92%.

Config	Test Accuracy (%)
PD-SGD	77.95
Interrupt	83.52
Matcha	85.72
FedCurv	86.03
MatchCurv	89.89

Table 6.11: The table shows the improvements in the moderate scenario, to the PDSGD with each of the individual solutions and final our proposed solution, MatchCurv.

On the other hand, in the extreme scenario, PD-SGD's performance drops drastically to 20%, making even random guessing of class labels appealing. When employing straggler interruption, Matcha, and FedCurv separately, we managed to improve it to 46%. Yet, MatchCurv still outperforms the rest, achieving a 54% accuracy. While it remains distant from ML performance, this highlights the importance of carefully selecting hyperparameters, regardless of the framework.
These experiments use a simple multi-class logistic regression model and the MNIST dataset for handwritten digit recognition. To further validate our framework's ro-

Config	Test Accuracy (%)
PD-SGD	20.66
Interrupt	30.86
Matcha	34.31
FedCurv	46.72
MatchCurv	54.54

Table 6.12: The table shows the improvements in the extreme scenario, to the PDSGD with each of the individual solutions and final our proposed solution, MatchCurv.

bustness, we also conducted the same experiments using a multi-layer perceptron (MLP) with two hidden layers of 128 units each. This time, we use the Fashion MNIST dataset for image classification, while keeping the other hyperparameters consistent. The results show a similar trend, with performance improving from 72% to 89% in the moderate case and from 19% to 44% in the extreme case compared to PD-SGD.

Chapter 7
Conclusion

In this thesis, we embark on a comprehensive exploration of decentralized federated learning (DFL), aiming to uncover its various intricacies and challenges. Our investigation spanned critical aspects of this evolving field, including decentralized optimization, efficient DFL communication strategies, and strategies to cope with statistical and system heterogeneity. Our objective was to address the existing gap in the literature—a lack of a robust DFL framework that is both communication efficient and capable of handling statistical and system heterogeneity.

We built upon existing work by merging two important frameworks: Matcha, which emphasizes efficient information sharing, and FedCurv, which introduces techniques to handle statistical heterogeneity through parameter penalization. It's worth noting that we carefully selected these frameworks to ensure they complemented each other effectively. Additionally, we incorporated the concept of accepting partial work from straggler devices into our framework. This further enhanced our approach's robustness and adaptability, allowing us to make the most of the resources available in a decentralized learning environment.

Through meticulously designed experiments conducted in diverse scenarios, we demonstrated the exceptional performance of our framework. We highlighted a significant performance gap between scenarios characterized by statistical heterogeneity and those conforming to the traditional IID (Independent and Identically Distributed) data settings. In this context, MatchCurv emerged as a potent solution, achieving accuracy levels comparable to IID scenarios while substantially reducing training time. Additionally, we explored the impact of various hyperparameters, network topologies, and the quality of network peers on system heterogeneity. Our experiments underscored MatchCurv's effectiveness by adding mechanisms to address stragglers, resulting in notable accuracy improvements even when dealing with devices of varying capabilities. Furthermore, we explored communication budgets inspired by the Matcha framework, illustrating how MatchCurv could optimize communication efficiency when integrated with Matcha and FedCurv.

In a comprehensive evaluation that considered both statistical and system heterogeneity as well as communication budgets, MatchCurv emerged as a robust and effective solution. It closely approximated the performance of traditional machine learning methods under moderate scenarios and surpassed PD-SGD in extreme cases, showcasing its adaptability and resilience. Importantly, these achievements extended beyond logistic regression and the MNIST dataset; we validated MatchCurv's effectiveness using a multi-layer perceptron and the Fashion MNIST dataset, consistently surpassing the performance of PD-SGD.

In summary, MatchCurv presents a powerful solution to the intricate challenges posed by statistical and system heterogeneity in DFL, all while optimizing communication efficiency. Its ability to enhance model accuracy and reduce training time positions it as a practical choice for decentralized learning environments. Our

overarching aim is to contribute to the development of a versatile DFL framework applicable across a wide array of machine learning domains. As we conclude this journey, we acknowledge that both opportunities and challenges lie ahead, and we aspire that our insights pave the way for further progress in the realm of decentralized federated learning.

7.1 Future Work

During the development of our framework, we encountered various design decisions and opportunities for improvement. In this context, we have identified several areas that require attention in relation to DFL and our framework. Firstly, we recognize the potential to enhance the security and efficiency of our system by incorporating established solutions like parameter compression and encryption. Secondly, we acknowledge that our current approach does not consider the possibility that model updates themselves could inadvertently reveal insights to a persistent adversary. We have assumed that all users are trustworthy, but it is essential to address this potential vulnerability. Differential privacy can be employed to prevent renegades from extracting sensitive information through model updates. Thirdly, we have not accounted for device dropouts during training, which should be handled more effectively for a robust framework. Lastly, exploring asynchronous training methods holds promise for further enhancing the performance and scalability of our framework.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Z. A. El Houda, A. Hafid, and L. Khoukhi, "Co-iot: A collaborative ddos mitigation scheme in iot environment based on blockchain using sdn", in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013542.

[2] K. Ochiai, K. Senkawa, N. Yamamoto, Y. Tanaka, and Y. Fukazawa, "Real-time on-device troubleshooting recommendation for smartphones", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2783–2791, ISBN: 9781450362016. DOI: 10.1145/3292500.3330669. [Online]. Available: https://doi.org/10.1145/3292500.3330669.

[3] M. Liaqat, V. Chang, A. Gani, *et al.*, "Federated cloud resource management: Review and discussion", *Journal of Network and Computer Applications*, vol. 77, pp. 87–105, 2017. [Online]. Available: https://eprints.soton.ac.uk/402161/.

[4] E. Parliament and of the Council of 27 April 2016, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance)", 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32016R0679.

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, 2023. arXiv: 1602.05629 [cs.LG].

[6] A. Hard, K. Rao, R. Mathews, *et al.*, *Federated learning for mobile keyboard prediction*, 2019. arXiv: 1811.03604 [cs.CL].

[7] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, *Federated learning for emoji prediction in a mobile keyboard*, 2019. arXiv: 1906.04329 [cs.CL].

[8] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, *Loadaboost: Loss-based adaboost federated machine learning with reduced computational complexity on iid and non-iid intensive care data*, 2020. arXiv: 1811.12629 [cs.LG].

[9] A. Pantelopoulos and N. G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 1, pp. 1–12, 2010. DOI: 10.1109/TSMCC.2009.2032660.

[10] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions", *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020. DOI: 10.1109/msp.2020.2975749. [Online]. Available: https://doi.org/10.1109%2Fmsp.2020.2975749.

[11] L. He, A. Bian, and M. Jaggi, *Cola: Decentralized linear learning*, 2019. arXiv: 1808.04883 [cs.DC].

[12] J. Huang, F. Qian, Y. Guo, *et al.*, "An in-depth study of lte: Effect of network protocol and application behavior on performance", *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 363–374, 2013, ISSN: 0146-4833. DOI: 10.1145/2534169.2486006. [Online]. Available: https://doi.org/10.1145/2534169.2486006.

[13] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, *The non-iid data quagmire of decentralized machine learning*, 2020. arXiv: 1910.00189 [cs.LG].

[14] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, *Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges*, 2023. arXiv: 2211.08413 [cs.LG].

[15] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, *Matcha: Speeding up decentralized sgd via matching decomposition sampling*, 2019. arXiv: 1905.09435 [cs.LG].

[16] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, *et al.*, "Overcoming catastrophic forgetting in neural networks", *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017. DOI: 10.1073/pnas.1611835114. [Online]. Available: https://doi.org/10.1073%2Fpnas.1611835114.

[17] N. Shoham, T. Avidor, A. Keren, *et al.*, *Overcoming forgetting in federated learning on non-iid data*, 2019. arXiv: 1910.07796 [cs.LG].

[18] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, *Federated optimization in heterogeneous networks*, 2020. arXiv: 1812.06127 [cs.LG].

[19] X. Li, W. Yang, S. Wang, and Z. Zhang, *Communication-efficient local decentralized sgd methods*, 2021. arXiv: 1910.09126 [stat.ML].

[20] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning", *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020. DOI: 10.1145/3377454. [Online]. Available: https://doi.org/10.1145%2F3377454.

[21] P. Zhou, Q. Lin, D. Loghin, B. C. Ooi, Y. Wu, and H. Yu, *Communication-efficient decentralized machine learning over heterogeneous networks*, 2020. arXiv: 2009.05766 [cs.DC].

[22] M. Oehlers and B. Fabian, "Graph metrics for internet robustness – a survey", 2021. DOI: 10.48550/ARXIV.2103.05554. arXiv: 2103.05554 [cs.NI].

[23] W. Liu, H. Sirisena, K. Pawlikowski, and A. McInnes, *Utility of algebraic connectivity metric in topology design of survivable networks*, 2009. DOI: 10.1109/drcn.2009.5340016.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[25] K. Fujita, K. Okada, and K. Katahira, *The fisher information matrix: A tutorial for calculation for decision making models*, 2022. DOI: 10.31234/osf.io/hdwut. [Online]. Available: psyarxiv.com/hdwut.

[26] R. Xin, S. Pu, A. Nedić, and U. A. Khan, *A general framework for decentralized optimization with first-order methods*, 2020. arXiv: 2009.05837 [cs.LG].

[27] T. Zhu, F. He, K. Chen, M. Song, and D. Tao, *Decentralized sgd and average-direction sam are asymptotically equivalent*, 2023. arXiv: 2306.02913 [cs.LG].

[28] H. Ouyang, N. He, L. Tran, and A. Gray, "Stochastic alternating direction method of multipliers", in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 80–88. [Online]. Available: https://proceedings.mlr.press/v28/ouyang13.html.

[29] L. Xiao, "Dual averaging method for regularized stochastic learning and online optimization", in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds., vol. 22, Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2009/file/7cce53cf90577442771720a370c3c723-Paper.pdf.

[30] Z. Yang and W. U. Bajwa, "ByRDiE: Byzantine-resilient distributed coordinate descent for decentralized learning", *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 4, pp. 611–627, Dec. 2019. DOI: 10.1109/tsipn.2019.2928176. [Online]. Available: https://doi.org/10.1109%2Ftsipn.2019.2928176.

[31] J. Zhang, Q. Ling, and A. M.-C. So, *A newton tracking algorithm with exact linear convergence rate for decentralized consensus optimization*, 2020. arXiv: 2008.10157 [math.OC].

[32] J. Liu, W. Chen, and H. Dai, "Event-triggered zero-gradient-sum distributed convex optimization over networks with time-varying topologies", *International Journal of Control*, vol. 92, pp. 1–27, Apr. 2018. DOI: 10.1080/00207179.2018.1460693.

[33] 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, Apr. 2014. DOI: 10.1109/tsp.2014.2304432. [Online]. Available: https://doi.org/10.1109%2Ftsp.2014.2304432.

[34] H. Li, Z. Lin, and Y. Fang, *Variance reduced extra and diging and their optimal acceleration for strongly convex decentralized optimization*, 2022. arXiv: 2009.04373 [math.OC].

[35] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, *Braintorrent: A peer-to-peer environment for decentralized federated learning*, 2019. arXiv: 1905.06731 [cs.LG].

[36] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges", *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4734–4746, 2020. DOI: 10.1109/TCOMM.2020.2990686.

[37] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, *Qsgd: Communication-efficient sgd via gradient quantization and encoding*, 2017. arXiv: 1610.02132 [cs.LG].

[38] Y. Lu and C. De Sa, "Moniqua: Modulo quantized communication in decentralized SGD", in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 13–18 Jul 2020, pp. 6415–6425. [Online]. Available: https://proceedings.mlr.press/v119/lu20a.html.

[39] T. Sun, D. Li, and B. Wang, *Decentralized federated averaging*, 2021. arXiv: 2104.11375 [cs.DC].

[40] C. Hu, J. Jiang, and Z. Wang, *Decentralized federated learning: A segmented gossip approach*, 2019. arXiv: 1908.07782 [cs.LG].

[41] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, *Federated multi-task learning*, 2018. arXiv: 1705.10467 [cs.LG].

[42] Q. Chen, Z. Wang, Y. Zhou, J. Chen, D. Xiao, and X. Lin, *Cfl: Cluster federated learning in large-scale peer-to-peer networks*, 2022. arXiv: 2204.03843 [cs.CR].

[43] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data", 2018. DOI: 10.48550/ARXIV.1806.00582. [Online]. Available: https://arxiv.org/abs/1806.00582.

[44] N. Onoszko, G. Karlsson, O. Mogren, and E. L. Zec, *Decentralized federated learning of deep neural networks on non-iid data*, 2021. arXiv: 2107.08517 [cs.LG].

[45] Y. Liao, Y. Xu, H. Xu, L. Wang, and C. Qian, *Adaptive configuration for heterogeneous participants in decentralized federated learning*, 2022. arXiv: 2212.02136 [cs.NI].

[46] J. Cao, Z. Lian, W. Liu, Z. Zhu, and C. Ji, *Hadfl: Heterogeneity-aware decentralized federated learning framework*, 2021. arXiv: 2111.08274 [cs.LG].

[47] E. Diao, J. Ding, and V. Tarokh, *Heterofl: Computation and communication efficient federated learning for heterogeneous clients*, 2021. arXiv: 2010.01264 [cs.LG].

[48] C. Lenzen and R. Wattenhofer, "Distributed algorithms for sensor networks", vol. 370, pp. 11–26, 2012, ISSN: 1364-503X. DOI: 10.1098/rsta.2011.0212.

[49] W. Wu, J. Cao, J. Yang, and M. Raynal, "A hierarchical consensus protocol for mobile ad hoc networks", vol. 2006, Mar. 2006, 9 pp. ISBN: 0-7695-2513-X. DOI: 10.1109/PDP.2006.11.

[50] S. Forster and D. Nanongkai, "A faster distributed single-source shortest paths algorithm", in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, Oct. 2018. DOI: 10.1109/focs.2018.00071. [Online]. Available: https://doi.org/10.1109%2Ffocs.2018.00071.

[51] F. Nourazar and M. Sabaei, "Dapf: An efficient flooding algorithm for mobile ad-hoc networks", in *2009 International Conference on Signal Processing Systems*, 2009, pp. 594–598. DOI: 10.1109/ICSPS.2009.112.

Appendix A

Appendix 1

In this section, we aim to provide the missing details regarding the framework and its evaluation. This setup involved conducting experiments in two different environments: Raspberry Pi devices and a simulation environment. It is noteworthy that the use of predefined random presets yielded consistent results across both environments, making it unnecessary to present them separately.
The MCLR model underwent 50 rounds of training in all of our experiments, where we explored the effects of statistical heterogeneity, systems heterogeneity, and communication budget. In these experiments, the number of epochs remained constant at 10 per training step, except when we investigated the impact of periodic communication. For all experiments, the learning rate was set to 0.01, and a batch size of 128 was utilized. These specific parameter values were determined through a grid search process, selecting the ones that yielded the best performance in conventional machine learning techniques.
Additionally, we include below some additional results that were omitted from the evaluation section.

Table A.1: Results with Different FedCurv Constants Under Various Non-IID Distributions Using MCLR Model and MNIST Dataset

Config	Curv Term λ	train_acc	test_acc
noniid2	0.01	0.964716	0.85725
noniid2	0.10	0.942461	0.82254
noniid2	1.00	0.920280	0.85556
noniid5	0.01	0.915115	0.91636
noniid5	0.10	0.908531	0.91222
noniid5	1.00	0.889981	0.89606

Table A.2: Impact of Stragglers and Interrupting them with MCLR Model on MNIST Dataset

Stragglers	Config	random_preset	train_acc	test_acc
0.25	interrupt	preset1	0.983035	0.776750
0.25	interrupt	preset2	0.983630	0.781520
0.25	interrupt	preset3	0.983320	0.781110
0.25	interrupt	preset4	0.983415	0.779710
0.50	interrupt	preset1	0.982493	0.786210
0.50	interrupt	preset2	0.982403	0.784080
0.50	interrupt	preset3	0.982179	0.780620
0.50	interrupt	preset4	0.982560	0.783560

Table A.3: Comparison of Results with Matcha Activations Using Different Presets with MCLR Model and MNIST Dataset

comm_budget	activations	random_preset	train_acc	test_acc
0.25	matcha	preset1	0.985219	0.45826
0.25	matcha	preset2	0.985568	0.63614
0.25	matcha	preset3	0.985162	0.64295
0.25	matcha	preset4	0.985147	0.69828
0.50	matcha	preset1	0.984587	0.73602
0.50	matcha	preset2	0.984500	0.72622
0.50	matcha	preset3	0.984279	0.75481
0.50	matcha	preset4	0.984537	0.73916

Table A.4: Analysis of FedCurv Constants in Different Scenarios with MCLR Model, MNIST Dataset, and PD-SGD Solver

Config	Curv Term λ	train_acc	test_acc
Extreme	0.01	0.983258	0.293860
Extreme	0.10	0.949992	0.360260
Extreme	1.00	0.922321	0.467160
Moderate	0.01	0.951979	0.830813
Moderate	0.10	0.933679	0.846500
Moderate	1.00	0.901205	0.860288

Furthermore, we would like to introduce a set of supplementary results derived from experiments conducted on the Fashion-MNIST dataset using the MLP model. It's worth noting that the network topologies employed here are identical to those used for training the MCLR model, and the hyperparameters remain consistent as well. The primary distinction lies in the dataset and the model architecture. In this case, the model consists of two hidden layers, each comprising 128 units.
In addition, we conducted a comprehensive evaluation of the robustness of our framework by subjecting it to experiments in a network comprising 100 devices or processes within the simulation environment. These experiments closely followed the methodologies outlined in the evaluation section, with the key distinction being the larger network scale. For this assessment, we utilized the standard MCLR model and trained it using the MNIST dataset, but this time with 100 devices. Below, we present the outcomes of these experiments:

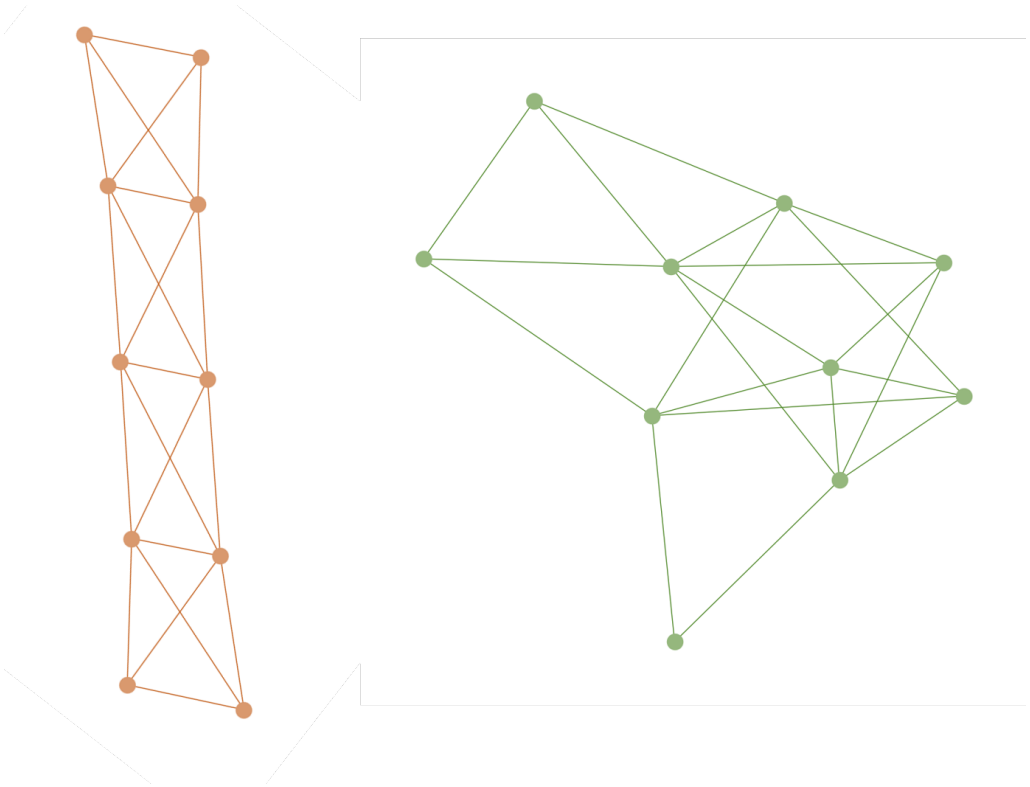


Figure A.1: The graph on the left consists of connected devices with highly similar data distributions, while the graph on the right, which has the same number of nodes and edges as the former, features diverse data distributions among neighboring devices.

Table A.5: Results with MCLR Model, MNIST Dataset, and MatchCurv

Curv Term λ	Curv Term λ	train_acc	test_acc
0.01	0.01	0.984404	0.38724
0.05	0.05	0.975757	0.40814
0.10	0.10	0.971022	0.42353
0.50	0.50	0.952441	0.49182
1.00	1.00	0.936113	0.52396
2.00	2.00	0.928746	0.54536
0.01	0.01	0.949240	0.89307
0.10	0.10	0.934838	0.89894
1.00	1.00	0.907184	0.89364

Config	Train Acc	Test Acc
Extreme	0.999740	0.199680
Moderate	0.998427	0.721088

Table A.6: Performance of PD-SGD in Moderate and Extreme Scenarios with MLP Model on Fashion MNIST Dataset.

Config	Train Acc	Test Acc
Extreme	0.999611	0.26851
Moderate	0.996423	0.80331

Table A.7: Results in Both Scenarios When Stragglers Are Interrupted with MLP Model on Fashion MNIST Dataset.

Config	Train Acc	Test Acc
Extreme	0.999871	0.338140
Moderate	0.998338	0.845437

Table A.8: Performance Results in Both Scenarios with Matcha on MLP Model and Fashion MNIST Dataset

Config	Curv Term λ	Train Acc	Test Acc
Extreme	0.01	0.999528	0.246940
Extreme	0.10	0.996737	0.284160
Extreme	1.00	0.988260	0.372880
Extreme	2.00	0.984509	0.376740
Moderate	0.01	0.991774	0.786525
Moderate	0.10	0.977129	0.812500
Moderate	1.00	0.950231	0.858313
Moderate	2.00	0.939996	0.865713

Table A.9: Performance Results in Both Scenarios with FedCurv on MLP Model and Fashion MNIST Dataset

Config	Curv Term λ	Train Acc	Test Acc
Extreme	0.01	0.999396	0.30501
Extreme	0.10	0.995909	0.35633
Extreme	1.00	0.986543	0.41226
Extreme	2.00	0.982571	0.44241
Moderate	0.01	0.990611	0.88408
Moderate	0.10	0.974845	0.89113
Moderate	1.00	0.946808	0.89272
Moderate	2.00	0.934647	0.89234

Table A.10: Accuracy Evaluation for Different Configurations with MatchCurv in Both Extreme and Moderate Scenarios with MLP Model on Fashion MNIST Dataset

Config	Train Acc	Test Acc
Extreme	0.990407	0.242134
Moderate	0.957954	0.880588

Table A.11: Results with PD-SGD in Both Scenarios Using MCLR Model, MNIST Dataset, and 100 Devices

Config	Curv Term λ	Train Acc	Random Preset	Test Acc
Extreme	0.01	preset1	0.981335	0.558969
Extreme	0.01	preset2	0.980124	0.518124
Extreme	0.10	preset1	0.968006	0.629453
Extreme	0.10	preset2	0.959255	0.598649
Extreme	1.00	preset1	0.932659	0.682916
Extreme	1.00	preset2	0.911250	0.639279
Moderate	0.01	preset1	0.922234	0.889396
Moderate	0.01	preset2	0.924202	0.890252
Moderate	0.10	preset1	0.909820	0.887057
Moderate	0.10	preset2	0.910632	0.887519
Moderate	1.00	preset1	0.889877	0.881265
Moderate	1.00	preset2	0.890584	0.882038

Table A.12: Results with MatchCurv in Both Scenarios Using MCLR Model, MNIST Dataset, and 100 Devices