# NUST COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING


DE-40(DEE)

# Decentralized Machine Learning

**A PROJECT REPORT**

DE-40 (DEE)

***Submitted by***

NC Haris Ghafoor

PC Awais Asghar

NC Malik Haseeb Haider

NC Haziq Ajam Malik

**BACHELORS**

**IN**

**ELECTRICAL ENGINEERING**

**YEAR 2022**

**PROJECT SUPERVISOR**

**Dr. Shahzor Ahmad**

**NUST COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING,**
**PESHAWAR ROAD, RAWALPINDI**

YEAR 2022

# Dedication

This project is dedicated to our parents, their efforts and all the prayers were with us during the project. It is also dedicated to our teachers and all the technical staff who really helped us for the completion of this project.

# Certificate of approval

It is to certify that the project "Decentralized Machine Learning" was done by NC Haris Ghafoor, PC Awais Asghar, NC Malik Haseeb Haider, NC Haziq Anjum Malik under the supervision of Dr. Shahzor Ahmed.

This project is submitted to **Department of Electrical Engineering**, College of Electrical and Mechanical Engineering (Peshawar Road Rawalpindi), National University of Sciences and Technology, Pakistan in partial fulfillment of requirements for the degree of Bachelors of Engineering in Electrical engineering.

**Students:**

**1- Awais Asghar**

NUST ID: 28079 Signature:

**2- Haris Ghafoor**

NUST ID: 264474 Signature:

**3- Malik Haseeb Haider**

NUST ID: 244931 Signature:

**4- Haziq Anjum Malik**

NUST ID: 258714 Signature:

**APPROVED BY:**

Project Supervisor: **Dr. Shahzor Ahmad** Date:

# Declaration

We hereby declare that no portion of the work referred to in this Project Thesis has been submitted in support of an application for another degree or qualification of this of any other university or other institute of learning. If any act of plagiarism is found, we are fully responsible for every disciplinary action taken against us depending upon the seriousness of the proven offense, even the cancellation of our degree.

**5- Awais Asghar**

NUST ID: 28079 Signature:

**6- Haris Ghafoor**

NUST ID: 264474 Signature:

**7- Malik Haseeb Haider**

NUST ID: 244931 Signature:

**8- Haziq Anjum Malik**

NUST ID: 258714 Signature:

# Copyright statement

# Acknowledgements

It is a great pleasure to write a report acknowledging efforts of many people whose names may not appear on the title page, but their hard work, friendship, cooperation, and understanding were with us to the production of this report.

Here we would like to mention the support and guidance given by our supervisor Dr. Shahzor Ahmed. His availability and prompt responses to our issues aided us in successfully finishing this job. Our department (Department of Electrical Engineering) is responsible for enabling us to complete this project and produce outcomes.

# Table of Contents

# Lists of Figures

# Abstract

Decentralized Machine Learning is the distributed training of an ML model on more than one device. It reduces the computational load on a device by distributing the centralized training models over many devices. Distributed training models, in contrast to centralized systems, utilize local datasets in a distributed network to ensure the confidentiality of user data. Large datasets boost accuracy, according to research, but training them takes a long time and a lot of computational power. As a result, decentralized machine learning is advantageous for applications that demand huge datasets, such as 3D scene generation, high image classification, and so on. There are two aspects of the problem: (1) Deploying ML models on edge devices and (2) distributed training of ML models. So our initial approach is to assume that both data and model are centralized and we need to decentralize the model by distributing the data across each device. Each device will be trained individually and will compute the model gradients. These gradients will be aggregated by the master/central server. The results of this approach were not satisfying as we observed disappointing results. After reviewing the recent work, we are working on devising the algorithm which will assign some initial weights to each client's gradients. Our initial investigation is that these effects impact the results strongly due to bias-shuffling of samples while distributing the dataset among the client/devices. So we can sum up the required research and development process as it involves (1) sending of the distributed datasets to the local devices (2) parallel training of each device using local datasets and (3) updating the hyper-parameters of local training models using dynamic model averaging.

# Introduction

Since 2016, artificial intelligence (AI) has progressed massively. We've seen the great potential of AI since it defeated the world's best human Go players, and more complicated, cutting-edge AI technology has become the norm in a range of areas, like autonomous automobiles, medical care, and finance, among others. AI is now being used in practically every sector of life.

However, when we look back on the evolution of AI, it is obvious that there have been some ups and downs. Will AI see another downturn? When will it appear, and why will it appear? The present public interest in AI is fueled in part by the availability of Big Data. Using a total of 300,000 games as training data, AlphaGo obtained good results. People instinctively expect that large data-driven Intelligence like Games will soon be realized in many facets of our life as a result of AlphaGo's success. However, Because most industries have limited or no data, we underestimated the complexity of AI technology..

Can we synthesize data from different companies in a single location? In many instances, breaking down the barriers between data sources is extremely difficult, if not impossible. In general, any AI project will require a variety of data kinds.

In a product suggestion service powered by AI, for example, the supplier of the service possesses product information and data, but no information about the user's purchasing skills or payment habits. In most sectors, data is distributed locally in data centers.

Sometimes data transformation across different departments in the same firm confronts substantial opposition due to market rivalry, privacy concerns, and complicated and expensive administrative regulations. It is either impossible or unreasonably expensive to integrate data from across the regions and organizations. Cyber-security has become a global concern as large organizations become more aware of the hazards of compromising the protection of data. The issue of data leaks has sparked widespread concern in the media and among governments. As a result, governments around the world are enacting new regulations to protect data security and privacy. The European Union's General Data Protection Regulation (GDPR) provides an example. GDPR is designed to safeguard users' personal information and data security. It compels businesses to use clear

and unambiguous language in their user agreements, and it gives users the "right to be forgotten," which allows them to remove and delete their personal data. Organizations that violate the law will face penalties.

For example, both China's Cybersecurity Law and General Principles of Civil Law state that Online businesses should not release or start messing with private information they collect, and when undertaking data communications with private entities, they must always ensure that one‘s main strategic planning meets the requirements with legitimate data security commitments [1].

These constraints will surely contribute to the development of more civil society, but they will also pose new challenges for The data transaction techniques that are currently used in AI. To be more specific, basic data transaction models are often used in traditional AI data processing models, in which one party gathers and sends data to another, with the latter party performing the clearing and integration of data. Consequently, private entities will use the stored data to create models that will be used by other parties. Models are typically finished items that are sold as a service. With the new data restrictions and laws mentioned above, this old approach confronts obstacles. Furthermore, because consumers may be unsure about the models' future uses, the transactions may be in violation of rules. We are faced with a conundrum in which our information is recorded in the form of isolated islands, but we are prohibited from collecting, fusing, and using the data in multiple places for AI processing in many instances. Today's AI researchers have a significant challenge in determining how to lawfully address data fragmentation and isolation. In this post, we'll look at a unique perspective called federated learning, which has the solution to these problems.

Federated learning is a well-known distributed learning framework that was created for edge devices. It allows an individual’s data to remain local while taking advantage of sizable processing provided by edge devices. In each federated, or communication, round, the purpose is to learn a shared model by alternating the following: Clients conduct numerous local changes after receiving a model from a server, while a server aggregates models from a subset of clients. Because DML systems often contain millions of edge devices, unknown diversity from multiple groups, restricted on-device capability, dynamic memory deployments, and inadequately annotated datasets, they are extremely hard to design [2] .

# Literature Review

## What is Federated Learning?

Define N data owners {F1, ...FN }, all of whom wish to train a machine learning model by consolidating their respective data {D1, ...DN }. A conventional method is to put all data together and use D = D1 ∪ ... ∪ DN to train a model $M_{SUM}$ . A federated learning system is a learning

process in which the data owners collaboratively train a model $M_{FED}$. In addition, it is expected that the performance of $M_{SUM}$ , $V_{SUM}$ and the performance of $M_{FED}$ , denoted as $V_{FED}$ should be very close. Let's assume that δ be a positive real number, if $| V_{FED} - V_{SUM} |$ < δ , it can be called the accuracy loss of the model [2].

Federated learning enables multiple parties to collaboratively construct a machine learning model while keeping their private training data private. As a novel technology, federated learning has several threads of originality, some of which are rooted in existing fields.

## Machine learning that protects your privacy

Federated learning and cross confidentiality machine learning are deeply linked because it is decentralized collaborative machine learning with privacy preservation. There has been a lot of research done in this field previously. For vertically segmented datasets, Vaidya and Clifton proposed protected association mining rules, secure k-means, and the Bayesian Classifier. An algorithm for data association rules with horizontal partitions. Algorithms for vertically and horizontally partitioned data are built using Secure Support Vector Machines. Multi-party linear regression and classification protocols that are secure. Multi-party gradient descent methods that are secure.

All of the following works used secure multi-party computing to ensure anonymity. Nikolaenko et al. used homomorphic encryption and Yao's garbled circuits to create a privacy-preserving linear regression protocol for horizontally partitioned data, as well as a linear regression technique for vertically partitioned data. These technologies provided a direct solution to the linear regression problem. They demonstrated privacy-preserving

algorithms for logistic regression and neural networks, as well as using Stochastic Gradient Descent to solve the challenge. A follow-up project with a three-server setup was just completed.

Aono et al. suggested a homomorphic encryption-based safe logistic regression algorithm. Shokri and Shmatikov proposed using updated parameters to train neural networks with horizontally partitioned data. To protect the privacy of gradients and improve the system's security, use additively homomorphic encryption. With recent advancements in deep learning, secured neural network inference is attracting a lot of study attention [3].

## Federated Learning vs Distributed Machine Learning

Horizontal federated learning appears to be comparable to distributed machine learning at first glance. Many areas of distributed machine learning are covered, including decentralized training storage space, distributed computing activities, and so on and distributed delivery of model results, among others. In distributed machine learning, a parameter server is a common component.

And to increase the speed of the training process and train the model more efficiently, the data is stored on multiple operating machines via the parameter server and it distributes data and computational resources through the master or central server. The data owner is represented by the working node in horizontally federated learning.

It has complete control over its local data and has the ability to decide when and how to engage in distributed learning. Federated learning encounters a more difficult learning environment since the central node frequently takes control of the parameter server. Second, during the model training process, federated learning prioritizes the data owner's privacy in the future, and effective data privacy protection measures will be better able to cope with a regulatory environment that is becoming increasingly strict in terms of data security and protection. Federated learning, like distributed machine learning, will have to deal with non-IID data[6]. In, it was shown that federated learning performance can be considerably degraded when using non-IID local data. In response, the authors proposed a novel strategy for dealing with the problem, comparable to transfer learning.

## Federated Learning vs Edge Computing

Federated learning,provides an invaluable mechanism for privacy and synchronization, can be thought of as an operating system for computing at the edge. A broad category of gradient-descent algorithms is used to train machine learning models. From a theoretical standpoint, they investigate the decentralized stochastic gradient efficiency bound and offer a control algorithm that calculates to minimize the loss function within a given resource budget; it's important to achieve the correct balance between local updates and parametric modeling aggregation.

## Federated Learning vs Federated Database System

Federated Database Systems are systems that administer the entire system by combining many database components. The federated database idea is used to enable interoperability with several databases. In a federated database system, database units are frequently stored in distributed storage and in reality, the data within every storage entity is heterogeneous. It shares many characteristics with federated learning in terms of data kind and storage. However, in the process of connecting with one another, there are no privacy protection methods in place in the federated database system,and the information system has total access to all separable datasets.Additionally, the distributed dbms concentrates on fundamental data activities like adding, removing, querying, and integrating, whereas distributed learning's aim is to construct a collaborative framework for each data owner built on the idea of preserving confidentiality, so that the data's diverse rights and regulations can benefit us.

# DECENTRALIZED DEEP LEARNING:

## Multi-Access Edge Computing

According to a Nokia annual report, the amount of telecommunications Internet - of - things and essential Internet of things devices, like Augmented reality technology, Virtual reality technology, and virtualized robotic systems, will greatly expand. Similarly, the number of huge IoT devices, such as various types of meters and sensors, is expected to skyrocket. Artificial Intelligence (AI) will be used to operate the majority of these devices. In wireless communications, Artificial Intelligence has been employed to

optimize edge services for the Internet of Vehicles (IoV) and other appealing collective intelligence applications. Traditionally, data created on a smart device is transferred to an external processing server for processing.

Though 5G seeks to provide better connectivity for a variety of devices as well as a significant increase in the speed with which large amounts of data can be handled, there is still a need for more coverage to allow for efficient data processing. As a result, combining MEC technology with latency reduction is preferable.

## Data Privacy and Decentralized Deep Learning

A probabilistic model for storing information, the perceptron mathematical model is used. Because of the multi-layer perceptron, neural networks are a more practical alternative to traditional statistical modeling techniques.

Deep learning (DL) is a technique for learning input representations at multiple levels of abstraction using computer models with multiple processing layers. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and other deep learning (DL) models have been developed and are now widely used in a variety of fields.

Distributed data processing topologies can be divided into two categories:

- centralized DL
- decentralized DL

A centralized model takes data from a range of resources and employs a central powerful computational capability to attain the expected performance of the model. In this situation, the acquired data is typically accessible to a cloud-based AI system. A distributed approach is considered a confidentiality architecture since it uses local model training based on heterogeneous datasets on limited capability devices like mobile-phones

The decentralized structure has spread throughout academia and industry since its inception. Li et al. developed a stable, adaptable, and high-performance implementation of the parameter server architecture, able to handle a variety of decentralized machine learning approaches based on local data sets. Furthermore, federated learning (FL), which was first proposed by Google to improve the performance of the Google Keyboard

(Gboard) in next word prediction, has become one of the most well-known decentralized frameworks in recent years. FL's architecture lets users fully utilize an AI algorithm without revealing their original local training data, thereby bridging the gap between centralized computer resources and dispersed data sources. By employing a globally shared model, FL achieves superior model parameters.

A completely decentralized machine learning framework, on the other hand, capable of adapting to server-less systems based on blockchains, edge consensus, and ad hoc networks. Swarm Learning (SL), for example, is a decentralized strategy for diagnosing diseases with dispersed medical data that integrates edge computing, peer-to-peer sharing on the smart contracts

The centralized data processing of a standard DL pipeline raises issues about data privacy, necessitating further thought into privacy-preserving system architecture and data security techniques. To that aim, the decentralized framework offers a viable data privacy solution for large-scale multi-agent collaborative learning. Industrial Io, environmental monitoring with a variety of sensors, from video surveillance to automation, social behavior is observe,. connected autonomous vehicle control, as well as multi-party decentralized network attacks detecting, for example, can all benefit from massively decentralized nodes [8].

## Federated Learning from a Network System Perspective

## Cross-functionality framework:

In FL's cross-silo configuration, data is acquired for computation, and locally resided data is sent to an internal edge server. In this process, a remote the server which is higher level is used to perform additional calculations. Within business, the Data is visible to all clients, but not to that outside of it. Healthcare institutions, for example, may use this approach to communicate medical photos in order to diagnose a rare disease. In this scenario, the cross-silo setup allows the institutes to share disease-related findings while maintaining data security. A cross-device setting, on the other hand, is a more stringent scenario in which data should not be allowed to leave a device. It demands effective on-device computation and model manner, transmission manner transmission to a remote server directly [3][10].

## Client Selection Policy:

To reduce waiting time, at the end of each cycle, the Parametric Processor chooses a limited set of k out of m clients randomly for training the localized model and transmits the current global model w to a selected client.

The Client selection procedures are also used to lower the time cost of global model convergence, cluster-based selection, and reinforcement learning-based selection are two examples of this type of selection.

## Synchronism:

We can categorize the scheduling of clients as:

- Synchronous FL
- Asynchronous FL.

Before going on to the next cycle in synchronous FL, the PS waits for all allotted local training to be finished. Because of large datasets, computing hardware restrictions, and other factors, the slowest local training task becomes the training bottleneck in this case.

On the other hand, the asynchronous FL provides the opportunity to clients to provide the local updates to the central server during any instance of the process. A client can also provide a wide range of services, such as local model training, network traffic transit, and so on.

## Aggregation:

In the next round's global model, the previously stated value of all local model updates' aggregate results is used. Because the amount of local training data differs per client, federated averaging (FedAvg) calculates a weighted average to update the global model (the contribution is varying) [2].

$$w_{t+1} = w_t + \sum_{i \in k} (n_i / n_k) (w^i{}_t - w_t) \quad (1)$$

Where wt is the current global model's weights, wt+1 is the next round's updated global model's weights, w is the client's trained local model's weights, and ni and nk are the volume of the client's local training data and the total training data from all the selected clients, respectively.

Furthermore, strong aggregation algorithms compare local model changes for similarity in order to detect a fraudulent update. At each round, only qualified updates are aggregated into the global model, based on the integrity of a local update.

## Deep Learning Models:

A supervised model is used in the majority of current FL research and implementations, in which the model is trained using labeled data for a classification goal. In practice, obtained data is frequently unlabeled, making a supervised model incompatible. Deep learning models like unsupervised learning and reinforcement learning have received little attention in the context of FL. For a robotics reinforcement problem, a global agent could, for example, use FL to learn various action policies from multiple settings at the same time.

## Client Server Network Security:

As a network system, FL is challenged by three system components: the parameter server, the client, and the transmission method. In comparison to edge devices, the PS is usually adequately guarded and regularly maintained. Furthermore, end-to-end encryption is typically used to protect communication between the PS and a client.

Even if a client's integrity is certified in order for them to participate in FL training, an edge still faces incursion from an adversary because its local defense methods are relatively insufficient. The adversary has a large attack surface to compromise the systems because all clients in FL have equal access to the global model via the aggregated model broadcast at each round. As a result, we believe the primary threat to FL systems is a compromised edge.

## Privacy of Federated Learning:

The privacy aspect of federated learning is one of its most important features. To give significant privacy guarantees, security models and analysis are required. This section examines and compares alternative privacy solutions for federated learning, as well as ways to limit indirect leakage and potential obstacles.

## Secure Multi-party Computation (SMC):

Different individuals are inherently involved in SMC security mechanisms that give secure proof with a well modeling tool with 100% having no idea. That implies every participant just understands its very own inputs. Although nil is a good quality, it typically requires complex computer techniques that are time-consuming to perform. In certain cases, limited information exchange could be appropriate if security requirements are met. In return for efficiency, it is to make sure to construct a secure model with SMC that has reduced security criteria. Recently, research employing two servers and semi-honest assumptions employed the SMC framework to train machine learning models. Users can train and verify models using MPC protocols without revealing sensitive data. One of the most advanced SMC frameworks on the market is Share Mind. With a 3-PC model and a true majority, consider security under both semi-honest and malevolent assumptions. Non-colluding servers must share participant data in secret in order for these projects to succeed.

## Differential Privacy:

Differential privacy or k-anonymity approaches are used in another line of work to protect data privacy. Differential privacy, k-anonymity, and diversification approaches entail introducing noise to the data or utilizing generalization methods to conceal certain critical qualities until a third party cannot recognize the individual, rendering the data unrecoverable and thereby protecting the user's privacy. The foundation of these methods still necessitates the transmission of data to a third party, and this effort frequently entails a trade-off between accuracy and privacy. A federated learning strategy that uses differential privacy to secure client-side data by masking clients' involvement during training.

**Homomorphic Encryption:**

Homomorphic Encryption is also used to secure client data privacy during machine learning by encrypting parameter exchange. Both the information and the models are not transferred in differential information privacy, whereas they can be inferred by other groups' data. There is a low risk of data leaking at the raw data level. Recently, homomorphic encryption has been used to centralize and train data on the cloud. In reality, Homomorphic Encryption is widely employed, yet ML algorithms need polynomial estimates to assess non-linear functions, which leads to inefficiency and security exchange.

**Indirect information leakage:**

There is no guarantee of security, and exposing intermediate results like weight updates from an optimization technique like SGD may actually release vital data information when combined with a data structure like image pixels. They also discussed numerous countermelodies, demonstrating that they may determine membership and qualities associated with a section of the training data from an antagonistic player, possibly security risk linked to gradient interactions between different parties.

## Privacy-preserving machine learning:

Distributed learning is closely connected to multi-party confidentiality learning algorithms, as it is a decentralized collaboration of machine learning which supports security. techniques to establish a secure inter tree structure for diagonally split data. Secret techniques are utilized for multiple linear regression supervised learning. Techniques of inter stochastic gradients that are safe.

Federated learning allows numerous parties to work together to build a machine learning model while maintaining the privacy of their training data. Federated learning, as a new technology, contains various original threads, some of which are founded in established domains. Shokri and Shmatikov proposed using updated parameters to train neural networks with horizontally partitioned data. To protect the privacy of gradients and improve the system's security, use additively homomorphic encryption. With recent developments in deep learning, privacy-preserving algorithms are now possible.

# Methodology

## Federated Averaging

There are many custom server aggregation algorithms available on TensorFlow. The main idea of server aggregation is to provide the aggregated weights which optimizes the local models. We can categorize the aggregation as:

- Applying Weighted Average when the locally distributed datasets are not identical in size or their datasets do not have balanced labels of all the classes. Those clients are given more importance who contribute more samples in the dataset. While other clients are multiplied by less than or equal to 0.5 coefficients to balance out the averaging [2].
- Applying Simple Averaging when the locally datasets are IID and balanced. It can be seen in the next chapter that these aggregated weights help local models to reach global minima.

Techniques in distributed optimizing, in particular, should deal with datasets that have the essential specifications:

• Massively Distributed: Large nodes K, are used to store sets of data. The node, in particular, can be considerably more than the mean amount of training data kept on each node (n/K).

• Non-IID: Every node's information could come from a unique distribution; in other words, the data points available locally tend unlikely to be a random sample of the total distribution.

• Unbalanced: The amount of training set held by various nodes might vary by orders of magnitude.

Sparsity data, where some attributes only appear on a tiny subset of nodes or data points, is of special relevance. While it was not a required feature of the federated optimization setup, we will demonstrate how the sparse framework may be utilized to design an efficient federated optimization method. It's important to note that the data produced by the major deep learning challenge presently being solved — predicting ad select rates —

is quite scarce.We're especially interested inside the situation wherein users' training data is stored on their smartphones and the data is highly confidential. This information xi, yi is created as a result of technology usage, such as application engagement. Identifying a next phrase a client will write (linguistic modeling for intelligent keyboard applications), forecasting what photographs a client would be most inclined to share, and forecasting which alerts are most essential all are instances of predictive analytics.To build these models utilizing typical distributed techniques, the training samples would've been collected at a centralized place (cloud server) then jumbled and spread evenly between different computing nodes. They suggest and investigate a different approach in which the training data really aren't delivered to a centralized place, possibly conserving network traffic and enhancing privacy and security.Clients agree to give up part of their product's computational power in return for the model's training. We transmit an update R d to a main system every round inside the communication system we utilize, whereby d is the dimensionality of the models getting approximated. As an instance, this updating may be a gradient vector. Although it is feasible for the may include significant confidential data about the user in some apps, it's indeed expected to be much less critical (and orders of magnitude less) than the source data. Think about the situation when the original training data consists of a vast collection of video recordings stored on a smartphone. Its feature's length will be unchanged by the size of the local training sample set. They demonstrate that even a globally model may be trained with only a few communication cycles, which decreases the network bandwidth required for training by orders of magnitude when comparing to transferring the information to a central. Because remote access may be restricted in a widely dispersed context, communication restrictions occur automatically . As a result, in practical settings, individuals may only be able to communicate once every day. Companies possess nearly infinite local processing capacity within tolerable constraints. As a result, the only practicable goal is to reduce the variety of communication cycles [7] . The work's major goal is to launch start investigation on distributed optimization and create the first effective application. Our results indicate that, also with right optimization techniques, n't yet getting an IID part of the data usable causes very little loss, and even with many nodes, we could still accomplish integration in some few cycles of communication. Google revealed recently that this notion has been implemented under one of their programmes, which has over 500 million users.

## Embedded systems & Edge ML:

Any piece of technology that regulates transmission of data just at the interface between two connections is referred to as an edge device. Regardless of the type of equipment, edge devices offer a range of functions, but they always act as network entrance — or exit — points. The transfer, transit, computing, observation, screening, translating, and storage of data moving through networks are all key tasks of edge devices. Organizations and service companies employ edge devices.

Edge Machine Learning (Edge ML) is a method of decreasing dependency on Cloud platforms by allowing the Significance Of having to analyze data locally (either utilizing local servers or at the device level) using machine and deep learning techniques. The word "edge" relates to both deep- and machine-learning techniques, which analyze data at the device's or local level (closest to the components collecting the data). Edge devices can transmit data to the Cloud as necessary, however, allowing users to handle certain data locally enables collected data while also enabling real-time data processing.

The research makes more use of the NVIDIA Jetson Nano component. As a result, the work's long-term viability is improved, and it may function on a real - time basis. All components of the Jetson Nano card box are shown in the diagram following. Along with software Etcher, the picture generated for Jetson Nano, which can be found on NVIDIA's official website,This images makes the CUDA and cuDNN libraries for Ubuntu Linux as well as NVIDIA products for the ARM64 processor available for usage.The deep learning model's data preparations should be completed following installing and operating the sd card on the Jetson Nano card. The which was before the set of data of the model to be trained was uploaded on to Jetson Nano card via USB stick after preceding installations have indeed been completed. Following this stage, any model may be executed with the supplied data set and the model's training process can begin. It can also be used with a pre-programmed model card [4] [9].

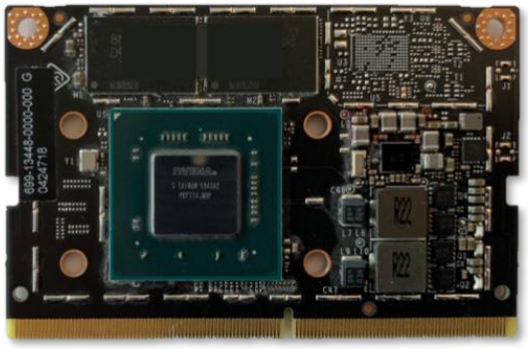

**Figure 1: Jetson Nano 4 GB RAM**

# Experimentation

In this section, we will discuss the details of the experiments performed. Our work can be divided into three parts:

- Implementation of **FedAvg** using **TensorFlow and** Comparison of results obtained by **Tensor flow's API** and our own decentralized model.
- Implementation of DML on different client's dataset
- Application on Boston Housing Dataset and EMNIST Dataset.

## TensorFlow's FedAvg and Limitations:

Researchers have used traditional MNIST training examples within that investigation to implement TFF's Federated Learning (FL) Application layer, tff. learning - a collection of relatively high integrations which can be used to function basic forms of federated instructional strategies, including federated training, against client specific models in TensorFlow. A federated training dataset, or data gathered from several users, is required for federated learning. Non-i.i.d. data is common with federated data, which presents a distinct set of issues. We populated our TFF library with some datasets to assist testing, along with a distributed update of MNIST, which includes a copy of the old MNIST data which has been re-processed utilizing Leaf such that the information is encrypted by the unique writer of the digits. Because each user does have a distinct style, this dataset demonstrates the non-IID behavior that federated datasets are known for. Distributed data is often non-i.d., and users' data distribution varies based on their usage habits. Certain users may well have fewer training sets on the devices due to a lack of data available, whilst others would have more than enough. Let's take a look at the notion of being in a federated system through using the EMNIST information we possess. It's essential to remember that this in-depth study of a customer's information is only available to us since we're in a simulation model with all the information at hand. In a real-world federated setup, inside an actual federated configuration, we will not be able to validate the data of a specific customer.

To begin, we took a sample of data from one user to have a feeling for such instances on a single imagined device. The input of one user reflects the handwritten of one individual for a sampling of the numbers 0 to 9, imitating the specific "usage behavior" of one user, because the dataset we're utilizing was keyed by a single writer.

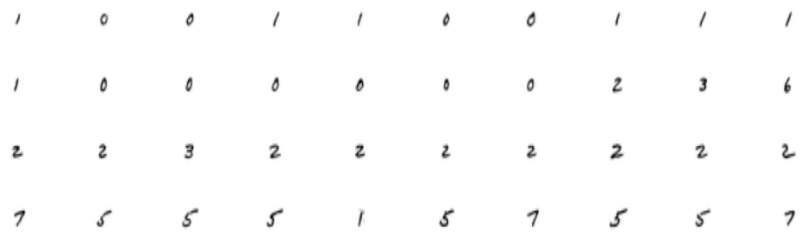

Figure 2: EMNIST Dataset's Visualization

They should see how many instances each MNIST character labeling has on every client. Depending on user activity, the number of instances on every server in a federated system might vary significantly.

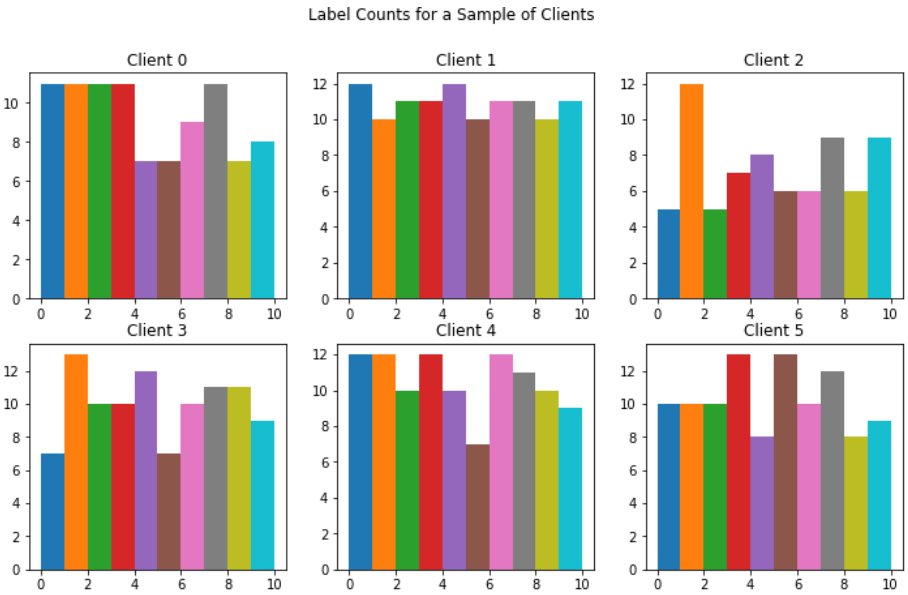


Figure 3: Insight of EMNIST Data

We compress all 28x28 photos to 784-element arrays, mix the various samples, sort these into groups, rebrand the attributes from pixels to x and y that can be used with Keras, We include a repetition over data in order to run many iterations.

```
NUM_CLIENTS = 10

NUM_EPOCHS = 5

BATCH_SIZE = 20

SHUFFLE_BUFFER = 100

PREFETCH_BUFFER = 10

def preprocess(dataset):

  def batch_format_fn(element):

    """Flatten a batch `pixels` and return the features as an
`OrderedDict`."""

    return collections.OrderedDict(

        x=tf.reshape(element['pixels'], [-1, 784]),

        y=tf.reshape(element['label'], [-1, 1]))


  return           dataset.repeat(NUM_EPOCHS).shuffle(SHUFFLE_BUFFER,
seed=1).batch(

      BATCH_SIZE).map(batch_format_fn).prefetch(PREFETCH_BUFFER)
```

We've got practically all the pieces in place to put together federated datasets.

Some other way to supply federation information to TFF in a model is as a Py lists, with each item carrying information about a specific client.

There must be two optimizers inside the Federated Averaging technique below:

- Client optimizer
- Server optimizer

The client optimizer is only used to calculate local model updates on each client. The server optimizer updates the global model by aggregating now at server level. This means that the optimization or learning rates you chose today can be different from what you used to build the machine on a normal i.i.d. dataset.

```
iterative_process = tff.learning.build_federated_averaging_process(
    model_fn,
    client_optimizer_fn=lambda:
tf.keras.optimizers.SGD(learning_rate=0.02),
    server_optimizer_fn=lambda:
tf.keras.optimizers.SGD(learning_rate=1.0))
```

TFF aims to design algorithms in such a manner that they'll be performed in actual federated learning situations, although at the moment, just localized implementation simulator runtime is available. You can use a simulator to run a calculation by calling it like a Python function. This basic interpretation runtime is not optimised for speed, but it will work for this lesson; in future editions, we plan to provide greater simulated runtimes to enable greater studies.

## The Centralized Model's Performance:

Even though the model in this instance is basic, there are a variety of ways to increase the performance of a model, such as utilizing more powerful models, extending iterations, or feature subset optimization. The objective is to compare the productivity of a dominant model formed by combining the attributes of localized models trained on their own data with an integrated process trained on all training data. This is how researchers can understand much more about the potential of distributed learning.

```
------ Centralized Model ------
epoch:   1 | train accuracy:  0.8725 | test accuracy:  0.9479
epoch:   2 | train accuracy:  0.9572 | test accuracy:  0.9663
epoch:   3 | train accuracy:  0.9710 | test accuracy:  0.9713
epoch:   4 | train accuracy:  0.9780 | test accuracy:  0.9715
epoch:   5 | train accuracy:  0.9833 | test accuracy:  0.9725
epoch:   6 | train accuracy:  0.9864 | test accuracy:  0.9780
epoch:   7 | train accuracy:  0.9900 | test accuracy:  0.9762
epoch:   8 | train accuracy:  0.9923 | test accuracy:  0.9811
```

```
epoch:   9 | train accuracy:  0.9933 | test accuracy:  0.9791
epoch:  10 | train accuracy:  0.9950 | test accuracy:  0.9779
                ------ Training finished ------.
```

## Distributing Data to Nodes:

```
abel_dict_train=split_and_shuffle_labels(y_data=y_train, seed=1,
amount=train_amount)

sample_dict_train=get_iid_subsamples_indices(label_dict=label_dict_train
, number_of_samples=number_of_samples, amount=train_amount)

x_train_dict, y_train_dict =
create_iid_subsamples(sample_dict=sample_dict_train, x_data=x_train,
y_data=y_train, x_name="x_train", y_name="y_train")
```

```
label_dict_valid = split_and_shuffle_labels(y_data=y_valid, seed=1,
amount=train_amount)

sample_dict_valid =
get_iid_subsamples_indices(label_dict=label_dict_valid,
number_of_samples=number_of_samples, amount=valid_amount)

x_valid_dict, y_valid_dict =
create_iid_subsamples(sample_dict=sample_dict_valid, x_data=x_valid,
y_data=y_valid, x_name="x_valid", y_name="y_valid")
```

```
label_dict_test = split_and_shuffle_labels(y_data=y_test, seed=1,
amount=test_amount)

sample_dict_test =
get_iid_subsamples_indices(label_dict=label_dict_test,
number_of_samples=number_of_samples, amount=test_amount)

x_test_dict, y_test_dict =
create_iid_subsamples(sample_dict=sample_dict_test, x_data=x_test,
y_data=y_test, x_name="x_test", y_name="y_test")
```

## Parameters of the main model are sent to nodes

Every one of these variables would be unique from one another, but since the main model's parameters and the parameters of all model parameters inside the nodes are randomly initialized. As just a result, even before the training of model parameters in the nodes starts, the primary model communicates its parameters to the nodes.

```
model_dict=send_main_model_to_nodes_and_update_model_dict(main_model,·m
odel_dict,·number_of_sampless
```

## Federated main model vs. individual local models before 1st iteration (on distributed test set)

The efficiency of the primary model is quite low because it is randomly initialized and also no action has been done on it yet. Maybe use before acc table as just a guideline.

```
before_acc_table=compare_local_and_merged_model_performance(number_of_sa
mples=number_of_samples)

before_test_loss, before_test_accuracy = validation(main_model, test_dl,
main_criterion)

main_model=
set_averaged_weights_as_main_model_weights_and_update_main_model(main_mo
del,model_dict, number_of_samples)


after_acc_table=compare_local_and_merged_model_performance(number_of_sam
ples=number_of_samples)

after_test_loss, after_test_accuracy = validation(main_model, test_dl,
main_criterion)
```

Federated main model vs individual local models before FedAvg first iteration

| | sample | local_ind_model | merged_main_model |
|---|---|---|---|
| 0 | sample 0 | 0.8111 | 0.0889 |
| 1 | sample 1 | 0.7667 | 0.1222 |
| 2 | sample 2 | 0.7111 | 0.1556 |
| 3 | sample 3 | 0.8111 | 0.1222 |
| 4 | sample 4 | 0.7222 | 0.1222 |

**Figure 4: Accuracy Table Before Ist Iteration**

Federated main model vs individual local models after FedAvg first iteration

| | sample | local_ind_model | merged_main_model |
|---|---|---|---|
| 0 | sample 0 | 0.8111 | 0.9000 |
| 1 | sample 1 | 0.7667 | 0.8222 |
| 2 | sample 2 | 0.7111 | 0.7889 |
| 3 | sample 3 | 0.8111 | 0.8556 |
| 4 | sample 4 | 0.7222 | 0.8667 |

**Figure 5: Accuracy Table after Ist Iteration**

We may transfer these weights of the primary model back to the nodes & continue the procedures above if that was a single iteration. And then see how the primary model's performance increases once we run this cycle ten times more.

```
for i in range(10):
model_dict=send_main_model_to_nodes_and_update_model_dict(main_model,
model_dict, number_of_samples)

    start_train_end_node_process_without_print(number_of_samples)

    main_model=
set_averaged_weights_as_main_model_weights_and_update_main_model(main_mo
del,model_dict, number_of_samples)

    test_loss, test_accuracy = validation(main_model, test_dl,
main_criterion)

    print("Iteration", str(i+2), ": main_model accuracy on all test
data: {:7.4f}".format(test_accuracy))
```

```
Iteration 2: main_model accuracy on all test data:  0.8973

Iteration 3: main_model accuracy on all test data:  0.9098

Iteration 4: main_model accuracy on all test data:  0.9173

Iteration 5: main_model accuracy on all test data:  0.9231

Iteration 6: main_model accuracy on all test data:  0.9295

Iteration 7: main_model accuracy on all test data:  0.9348

Iteration 8: main_model accuracy on all test data:  0.9370

Iteration 9: main_model accuracy on all test data:  0.9393

Iteration 10: main_model accuracy on all test data:  0.9427

Iteration 11: main_model accuracy on all test data:  0.9441
```

The centralized model's efficiency was estimated to be around 98 percent. The accuracy of the primary model developed using the FedAvg approach increased from 85% to 94 percent. For instance, regardless of the fact that the FedAvg technique's fundamental model was developed without observing the data, its efficiency can really be dismissed.

## Decentralized Machine Learning On Boston Housing Dataset:

Our approach that can be matched to the TensorFlow API produced these findings. This dataset covers housing statistics gathered by us Census Service in the Boston, Massachusetts region. It was obtained from the StatLib archive (http://lib.stat.cmu.edu/datasets/boston) .Such analyses, though, were conducted mostly outside Delve and are hence doubtful. With only 506 instances, the dataset is tiny. Without spreading the dataset to any nodes, we generated a neural network for the entire dataset.

We have to be mindful not to develop an extremely complicated model due to the minimal quantity of supplied data in our dataset, since this might lead to generalizing our data. For this, we'll use a Dense layer design with two Dense layers, one with 128 neurons and the other with 64 neurons, all with Activation functions. The output layer is a dense layer with linear activity. We used a mean square loss function to determine if our model is effectively learning, and we will utilize the mean average error metric to quantify its

effectiveness. We could say we've got a maximum of 10,113 parameters utilizing Keras' overview approach, which is fine for us.

```
from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(128, input_shape=(13, ), activation='relu',
name='dense_1'))

model.add(Dense(64, activation='relu', name='dense_2'))

model.add(Dense(1, activation='linear', name='dense_output'))


model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

The results of the centralized model is as follows:

```
[ ]  mse_nn, mae_nn = model.evaluate(X_test, y_test)

     print('Mean squared error on test data: ', mse_nn)
     print('Mean absolute error on test data: ', mae_nn)

     5/5 [==============================] - 1s 156ms/step - loss: 18.7784 - mae: 2.7856
     Mean squared error on test data:  18.778406143188477
     Mean absolute error on test data:  2.7855632305145264
```

**Figure 6: Mse and Mae on Test data**

For the decentralized model, data was equally distributed to the available clients according to an independent and identically distributed random distribution. The pipeline of the project from the client's perspective is as follows:

- Step 1: Weights Initialization with count=0
- Step 2: Obtaining Weights and Sending them for Averaging
- Step 3: Waiting for until we pull the averaged weights
- Step 4: Setting the weights and fitting the model again
- Step 5: Repeat 2 to 4 until the model reaches the global minima or performance cannot be performed.

These Following definitions will explain the functions used in the client code:

**pre_processing(df):**

**Input:** This function takes a raw dataset as an input. The DataType of the dataset is the pandas DataFrame.

**Output:** The output is the processed or cleaned splitted training and testing lists which is done by exploratory data analysis and statistical operations to remove outliers and anomalies.

**Objective:** The main objective of this function is to clean the dataset which can be then fed into the neural network for the training/inference.

**Return Data Type**: The return datatype of all the variables are listed.

**unpickling_weights(filename):**

**Input:** This function takes json's filename as an input.

**Output:** The output is the list of weights and biases received from the server.

**Objective:** The main objective of this function is to load the updated weights received from the server.

**Return Data Type**: The return datatype is the list.

**Training_evaluating(X_train,y_train,X_test,y_test,count,features,json1=None,model_name=None,weight_file_name=None)**

**Input:** This function takes training and testing lists.Moreover, it also takes in the count value which is very important. The variable 'feature' decides the 1st layer shape of the neural network. And it also takes in updated weights jsons as an input.

**Output:** The output is just the printed variables like mae and maps or accuracy.

**Objective:** This function performs both training and evaluation for each round of communication. When the value of count = 0, the model starts training using randomly initialized weights. When the value of count >1, the model trains on the updated weights received by the central server.

**sending_models_and_weights(model,model_name,weight_file_name)**

**Input:** This function takes multiple strings (filenames) and model architecture pickled in the form of json.

**Output:** None

**Objective:** The main objective of this function is to send the locally updated weights to the server after retraining the model.

**Return Data Type**: None

**main_function(count)**

**Input:** This function takes the value of count which describes the round of communication between the local machine and the server.

**Output:** None

**Objective:** The main objective of this function is to train and evaluate the model using local optimization and global updates.

**Return Data Type**: None

**server_aggregation()**

This function is implemented on the server and only called once it receives all the weights from local devices. The important thing to note here is that here the dataset satisfies IID properties so that we did not have to use weighted averaging.

**Input:** This function takes the local weights in form of h5/jsons.

**Output:** The output is the weights and biases in the form of jsons.

**Objective:** The main objective of this function is to aggregate the local weights so that model can be optimized on a global dataset irrespective of the fact that it has not seen the data from other local devices.

**Return Data Type**: Updated Weights and Bias lists .

```
def server_aggregation():

    model_1=load_model(filename='emnist_client1.json')

    model_1.load_weights('emnist_client1.h5')

    model_2=load_model(filename="emnist_client2.json")

    model_2.load_weights('emnist_client2.h5')

    model_3=load_model(filename="emnist_client3.json")

    model_3.load_weights('emnist_client3.h5')

    averages=[
[[np.mean(np.array([l1.get_weights()[0],l2.get_weights()[0],l3.get_weigh
ts()[0]]),axis=0)],[np.mean(np.array([l1.get_weights()[1],l2.get_weights
()[1],l3.get_weights()[1]]),axis=0)]]       for       l1,l2,l3       in
zip(model_1.layers,model_2.layers,model_3.layers)]

    weight=[i[0][0]for i in averages]

    bias=[i[1][0] for i in averages]

    # Saving the Updated Weights and Sending the updates to Clients
Models:



    ls_layer_1=[]

    ls_layer_1.append(weight[0]) # Weights of layer 1

    ls_layer_1.append(bias[0]) # Bias of layer1




    ls_layer_2=[]

    ls_layer_2.append(weight[1]) # Weights of layer 1

    ls_layer_2.append(bias[1]) # Bias of layer1




    ls_layer_3=[]

    ls_layer_3.append(weight[2]) # Weights of layer 1

    ls_layer_3.append(bias[2]) # Bias of layer1
```

```
saving_updated_weights(ls_layer_1,ls_layer_2,ls_layer_3)
return ls_layer_1,ls_layer_2,ls_layer_3,
```

## Testing and Evaluation Results:

The above explained algorithm was applied on two decentralized clients. The results obtained by clients are as expected and the final optimized models gave better results than the centralized model. Following are the plots of the clients vs the round of communication.

From the plot of client 1, it can be observed that the final mae is less than the mae of the centralized model. While client 2 does not give as best results as obtained from the centralized model, this is due to unique and random distribution of dataset and their underlying unique patterns.

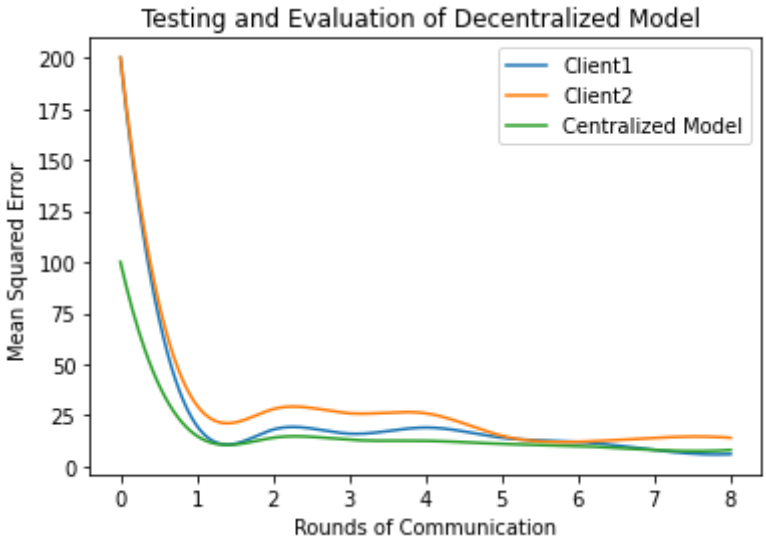


**Figure 7: Testing of DML on Boston Housing Dataset**

## Comparison of Centralized Model vs Decentralized Model

These are the results which we obtained after the final round of communication.

| Evaluation Metrics | Centralized Model | Client 1 of DML | Client 2 of DML |
|---|---|---|---|
| **Mean Absolute Error** | 2.53 | 1.99 | 3.006 |
| **Mean Squared Error** | 18.77 | 8.20 | 14.6 |
| **Time Elapsed** | 10s | 4.56 s / round | 3.77 s / rounds |

## Decentralized Machine Learning on MNIST Dataset:

The preceding chapter explains how DML was successfully implemented for a supervised regression task. In this topic, we will discuss the implementation of DML for classification problems and will compare the results as compared to the centralized model. This DML script was tested for three available clients.

For starters, the EMNIST dataset is a collection of handwritten character digits extracted from EMNIST Special Collection 19 and transformed to a 28x28 frame JPG format with such dataset architecture identical to the MNIST dataset.

The deep neural network architecture is defined as:

```
model = Sequential()

      model.add(Dense(128,     input_shape=(features,     ),
activation='sigmoid', name='dense_1'))

      model.add(Dense(64,              activation='sigmoid',
name='dense_2'))

      model.add(Dense(10,              activation='softmax',
```

```
name='dense_output'))


model.compile(optimizer='adam',loss='sparse_categorical_cros
sentropy',metrics=['accuracy'])

      model.fit(X_train, y_train, epochs=5)

      model.summary(
```

The rest of the pipeline is the same as it was in the case of regression. The sequences of the steps are as follows:

- Step 1: Weights Initialization with count=0
- Step 2: Obtaining Weights and Sending them for Averaging
- Step 3: Waiting for until we pull the averaged weights
- Step 4: Setting the weights and fitting the model again
- Step 5: Repeat 2 to 4 until the model reaches the global minima or performance cannot be improved further.

## Testing and Evaluation Results:

The above explained algorithm was applied on three decentralized clients. The results obtained by clients are as expected and the final optimized models gave better results than the centralized model. Following are the plots of the clients vs. the round of communication.

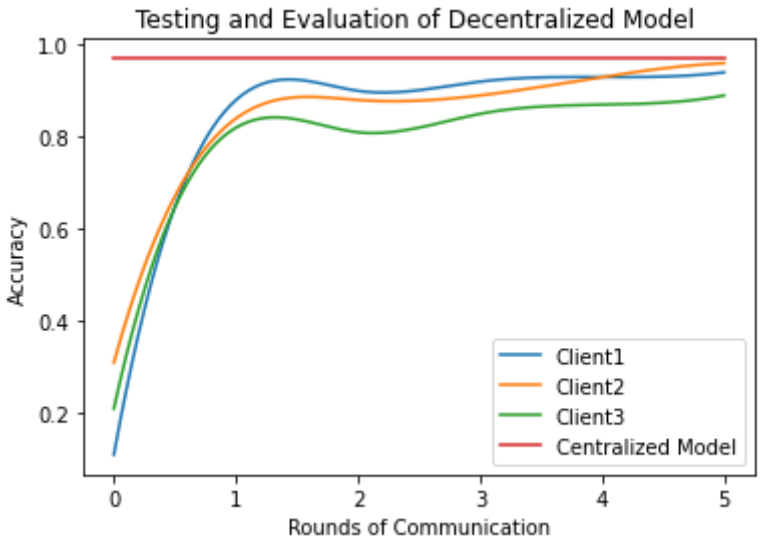


**Figure 8: Testing of DML on EMNIST Dataset**

## Comparison of Centralized Model vs. Decentralized Model

These are the results which we obtained after the final round of communication.

| **Evaluation Metrics** | **Centralized Model** | **Client 1 of DML** | **Client 2 of DML** | **Client 3 of DML** |
|---|---|---|---|---|
| **Accuracy** | 97 % | 94 % | 95 % | 89 % |
| **Time Elapsed** | 1 min | 30 s / round | 28 s / round | 30 s / round |

# Future Work

In the above sections, we explained that our algorithm assumes that locally distributed datasets are labeled in the form of X and Y where X is the training data while Y is the label of training data. But we cannot ignore the fact that there might be scenarios where data might not be available in the standard form due to many possible reasons. These challenging scenarios arise from collection obligations and limited resources. For example Medical Centers can easily use DML without sharing sensitive data of clients but there might be medical centers who don't have the resources to label the data or organize it in the standard form. So we need to explore the issues which deal with partially labeled X. partially labeled Y and irregular X respectively.

## Data incompleteness:

Clients can't see all of the realistic labels in many FL implementations. For example, a medical facility may seek to employ FL to increase diagnostic performance while avoiding the transmission of confidential documents from clinics located in remote locations. Clinics, on the other hand, lack modern healthcare capabilities to categorize all or most of the data. Some recent research on semi-supervised learning in a centralized environment already has demonstrated the enormous possibilities of using unsupervised learning. An actual case of special relevance among the partially-labeled FL setups is the "fully unlabeled clients," yet the server may contain some labeled data. To return to the scenario, a health center would have a few labels due to its extensive facilities, whereas distributed facilities would only have unlabeled data. Recent semi-supervised FL studies have revealed encouraging initial findings, indicating that with appropriately designed stability regularization techniques, performance comparable to centralized and fully-labeled training might be achieved with unlabeled users [5].

## Data polarity

Only data that contains a recognized keyword is stored by a computer based on natural language processing for further recognition, thus only positive data is gathered while negative sample is eliminated.

In such circumstances, the gathered training dataset does not reflect the entire distribution of the data, which might lead to significant prediction biases.

# Conclusion

Localization of data and a focus on consumer privacy have become the next obstacles for AI technology in recent years, but DML has given us fresh opportunities. It may create a unified paradigm for various businesses while keeping local information safe, allowing businesses to compete with us on the basis of data privacy. This project analyzes the possibilities of DML in numerous applications and covers the core concept, design, and methodologies of federated learning. In the coming years, federated learning is projected to break down boundaries across businesses and create a network where data and expertise may be exchanged safely while the rewards are shared evenly based on each user's contribution. AI benefits will eventually achieve all aspects of modern life.

Several complex and fascinating scientific questions remain to be investigated, as we mentioned throughout the study. The difficulties of information exchange among rounds, customization, absence of categories, resilience, and prolongation in DML are not well investigated from such an algorithmic perspective. Moreover, the issues of limitations at the edge devices, privacy, and confidentiality are all important topics from a system design perspective. FL research is quickly progressing in application areas outside computer vision and computational linguistics in parallel to computational and system-design challenges.

In the pharmaceutical research, social network, recommendation system, and advertisement domains, graph-structured data and time-series data are examples.

# References


1. A. Elgrabli, J. Park, A. S. Bedi, M. Bennis and V. Aggarwal, "Communication Efficient Framework for Decentralized Machine Learning," 2020 54th Annual Conference on Information Sciences and Systems (CISS), 2020, pp. 1-5, doi: 10.1109/CISS48834.2020.1570627384.
2. Tianyi Chen, Georgios Giannakis, Tao Sun, and Wotao Yin. Lag: Lazily aggregated gradient for communication-efficient distributed learning. Advances in Neural Information Processing Systems, 31:5055–5065, 2018.
3. S. S., J. I. Zong Chen, and S. Shakya, "Survey on Neural Network Architectures with Deep Learning," Journal of Soft Computing Paradigm, vol. 2, no. 3, 2020, doi: 10.36548/jscp.2020.3.007.
4. D. C. Nguyen et al., "Federated Learning Meets Blockchain in Edge Computing: Opportunities and Challenges," IEEE Internet of Things Journal, vol. 8, no. 16, pp. 12806–12825, 2021, doi: 10.1109/JIOT.2021.3072611.
5. A. Qayyum, K. Ahmad, M. A. Ahsan, A. Al-Fuqaha, and J. Qadir, "Collaborative Federated Learning For Healthcare: Multi-Modal COVID-19 Diagnosis at the Edge," Jan. 2021.
6. F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning from Non-IID Data," Mar. 2019.
7. Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, and D. Niyato, "Federated learning for 6g communications: Challenges, methods, and future directions," China Communications, vol. 17, no. 9, pp. 105–118, 2020.
8. Abadi, M. (2015), 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems', Software available from tensorflow.org
9. Loukadakis, M., Cano, J. and O'Boyle, M. (2018), 'Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures', 11th Int. Workshop on Programmability and Architectures for Heterogeneous Multicores.
10. L. Chu, L. Wang, Y. Dong, J. Pei, Z. Zhou, and Y. Zhang. Fedfair: Training fair models in cross-silo federated learning. arXiv preprint arXiv:2109.05662, 2021.

# Appendices

## A: Generic Client's Code

```
from flask import Flask, request,jsonify
import pickle
import os
import tensorflow as tf
import keras
from keras.initializers import glorot_uniform
from werkzeug.utils import secure_filename
import requests
import json
from sklearn.datasets import load_boston
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
from time import sleep
#from sklearn.datasets import load_boston
#from sklearn.model_selection import train_test_split
import pickle
import tensorflow as tf
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
#from sklearn.model_selection import train_test_split
#from sklearn.cross_validation import train_test_split
import pickle
import os
import tensorflow as tf
```

```
import winsound

def shuffle_split_data(X, y):
    arr_rand = np.random.rand(X.shape[0])
    split = arr_rand < np.percentile(arr_rand, 70)

    X_train = X[split]
    y_train = y[split]
    X_test =  X[~split]
    y_test = y[~split]

    print (len(X_train), len(y_train), len(X_test), len(y_test))
    return X_train, y_train, X_test, y_test



def pre_processing(df):
    df = df.iloc[:250,:]
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]
    X_train,y_train, X_test, y_test = shuffle_split_data(X, y)

    mean = X_train.mean(axis=0)
    std = X_train.std(axis=0)

    X_train = (X_train - mean) / std
    X_test = (X_test - mean) / std

    return X_train,y_train,X_test,y_test



def unpickling_weights(filename):
  with open(filename, "rb") as fp:
```

```
    ls=pickle.load(fp)

  return ls



def
Training_evaluating(X_train,y_train,X_test,y_test,count,features,json
1=None,model_name=None,weight_file_name=None):

    # global count

    if count==0:

      model = Sequential()

      model.add(Dense(128,          input_shape=(features,          ),
activation='relu', name='dense_1'))

      model.add(Dense(64, activation='relu', name='dense_2'))

      model.add(Dense(1, activation='linear', name='dense_output'))

      model.compile(optimizer='adam', loss='mse', metrics=['mae'])

      model.summary()


      history     =     model.fit(X_train,     y_train,     epochs=100,
validation_split=0.05)


      mse_nn, mae_nn = model.evaluate(X_test, y_test)


      print('Mean squared error on test data: ', mse_nn)

      print('Mean absolute error on test data: ', mae_nn)

      print('count: ', count )


      sending_models_and_weights(model,model_name,weight_file_name)

      count=1

    else:

        ls=unpickling_weights(json1)

        ls1=ls[0]

        ls2=ls[1]

        ls3=ls[2]


        with open(model_name, 'r') as json_file:
```

```
            json_savedModel= json_file.read()
        #json_savedModel
        #load the model architecture
        model = tf.keras.models.model_from_json(json_savedModel)
        model.summary()
        model.layers[0].set_weights(ls1)
        model.layers[1].set_weights(ls2)
        model.layers[2].set_weights(ls3)

        model.compile(optimizer='adam', loss='mse', metrics=['mae'])
        history    =    model.fit(X_train,    y_train,    epochs=100,
validation_split=0.05)

        mse_nn, mae_nn = model.evaluate(X_test, y_test)

        print('Mean squared error on test data: ', mse_nn)
        print('Mean absolute error on test data: ', mae_nn)

        print('count: ', count )
        count+=1

        #return model,history,mse_nn,mae_nn,count

def sending_models_and_weights(model,model_name,weight_file_name):


    # serialize model to json
    json_model = model.to_json()
    #json_model
    #save the model architecture to JSON file
    with open(model_name, 'w') as json_file:
        json_file.write(json_model)
```

```
    model.save_weights(weight_file_name)


count=0


def main_function(count):
    boston_dataset = pd.read_csv('BostonHousing.csv')
    df=boston_dataset
    X_train,y_train,X_test,y_test=pre_processing(df)
    rows,features=X_test.shape
    json1="updated_weights.json"
    model_name="client2_nn.json"

Training_evaluating(X_train,y_train,X_test,y_test,count,features=13,j
son1="updated_weights.json",model_name="client2_nn.json",weight_file_
name="client2_nn.h5")



donee= {}



def load_model(filename):
    # Creating a new model from the saved JSON file
    # reda the model from the JSOn file
    with open(filename, 'r') as json_file:
        json_savedModel= json_file.read()
    #json_savedModel
    #load the model architecture
    model = tf.keras.models.model_from_json(json_savedModel)
    model.summary()
    return model




app = Flask(__name__)
```

```
@app.route("/")
def hello_world():
    return request.remote_addr



@app.route("/initialise")
def intitialise():
    global count
    ipaddress_server= request.remote_addr
    main_function(count)
    data=open("client2_nn.h5","rb")
    jsonn=open("client2_nn.json","rb")

    dataaatogether={"data":data ,
                    "json":jsonn}
    # print(hex(jsonn))
    # print((jsonn))
    # print(jsonn)
    # sleep(10)
    if count<7:

asd=requests.get(url="https://"+ipaddress_server+":5000/recvweights",
files=dataaatogether,verify=False)
    count=count+1


    return "a"




@app.route("/client_recieve_updated",methods=['GET', 'POST'])
def client_recieve_updated():
    global count
```

```
    ipaddress_server= request.remote_addr
    dataa=request.files["data"]
    dataa.save(secure_filename(dataa.filename))
    main_function(count)
    # ipaddress_server


    data=open("client2_nn.h5","rb")
    jsonn=open("client2_nn.json","rb")


    dataaatogether={"data":data ,
                    "json":jsonn}
    # print(hex(jsonn))
    # print((jsonn))
    # print(jsonn)
    if count < 7:

asd=requests.get(url="https://"+ipaddress_server+":5000/recvweights",
files=dataaatogether,verify=False)


    count=count+1
    winsound.Beep(frequency=2500, duration=1000)


    return "a"




if __name__ == '__main__':
    app.run(host="0.0.0.0",port=5000,ssl_context="adhoc")
```