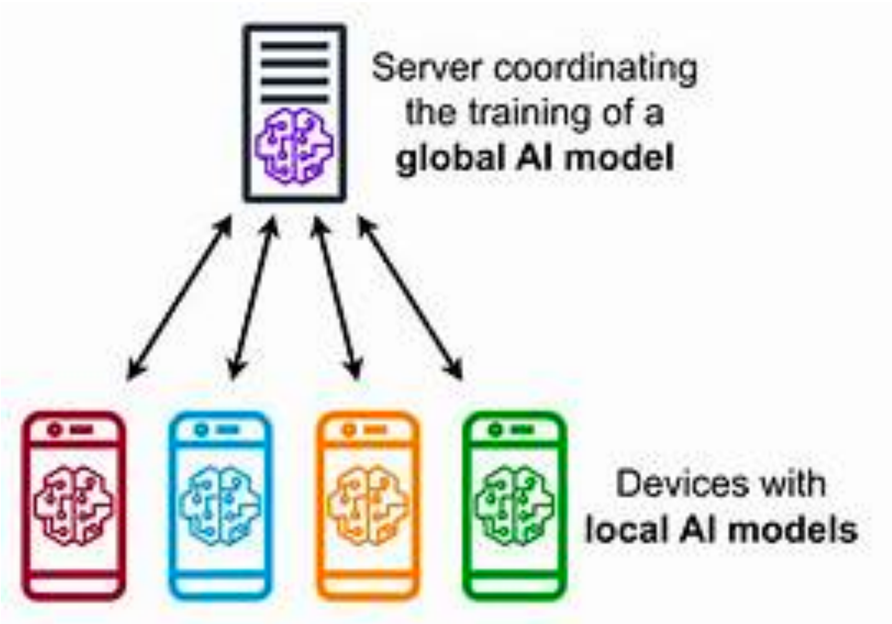
# Federated Learning for Privacy-Preserving Machine Learning

## Final Project Presentation

Raj Kumar Maurya, 2024.05.02

# Introduction

- Federated Learning (FL) is a decentralized machine learning approach where model training is performed across multiple edge devices or servers holding local data, without exchanging them. This project outlines the implementation of Federated Learning in a real-world scenario.



# Federated Learning Algorithms

- **Federated Averaging (FedAvg):** It is one of the foundational federated learning algorithms. In FedAvg, each participating device trains its local model on its own data, and then the model updates are aggregated (averaged) at a central server. This process helps in preserving data privacy since raw data doesn't leave the devices.
- **Federated Stochastic Gradient Descent (FedSGD):** Similar to traditional SGD, FedSGD optimizes a global model by updating parameters using gradients computed locally on each device. These local gradients are then aggregated to update the global model.
- There are many other algorithms:
  - Federated Momentum (FedMo)
  - Federated Proximal (FedProx)
  - Federated Learning with Adaptive Gradient Clipping (FLAG)
  - Federated Learning with Differential Privacy (FLDP)
  - Federated Learning with Differential Privacy (FedDP)
  - Federated Learning with Secure Aggregation (FedSecAgg)
  - Federated Learning with Homomorphic Encryption (FedHomEnc)

# Objective

- **Understand** Federated Learning: Study the principles, challenges, and benefits of federated learning.
- **Implement** Federated Learning techniques to leverage distributed data for model training.
- **Demonstrate** the feasibility of Federated Learning in improving model performance while maintaining data privacy.
- **Apply** federated learning algorithms to a specific use case and do comparative analysis

# Methodology

- **Literature Review:** Survey existing research on federated learning
- **System Architecture:** Design the system to implement the federated learning in a simulated environment or actually distributed systems
- **Algorithm Selection:** Choose appropriate federated learning algorithms based on use case requirements.
- **Implementation:** Develop a proof-of-concept system. Dataset and the problem statement to be finalised.
- **Evaluation Metrics:** Define metrics to assess model performance, communication efficiency, and privacy preservation.

# FedAvg vs FedSGD

| Aspect | FedSGD | FedAvg |
| --- | --- | --- |
| Aggregation Method | Aggregates gradients | Aggregates model parameters |
| Update Strategy | Updates global model directly with gradients | Averages model parameters to update global model |
| Communication Overhead | Higher due to gradient transmission | Lower due to parameter transmission |
| Privacy Preservation | Gradients may contain more information about local data | Averaged parameters may preserve privacy better |
| Frequency of Synchronization | Every iteration | After several local updates |

# Expected Outcomes

- **Functional Prototype:** A working federated learning system demonstrating privacy-preserving model training.
- **Performance Evaluation:** Comparative analysis of the two federated learning algorithms chosen.
- **Documentation:** Detailed project report, code snippets, and experimental results.

# Feedbacks from 1st Presentation

- *Focus on the distributed aspects*
- *Try with smaller models*
- *Use cluster of VMs provided instead of simulating locally*
  - *Since VM was not accessible over VPN, went ahead with simulation on multi-core system after getting confirmation from the TA*

# Solution Overview & Summary

- I have implement FedAvg and FedSGD Federated Learning technique on a **synthetic data** for a **regression problem** using **2 Layers Vanilla Neural Network** that highlights the properties of these algorithms.
- I have ran **various experiments** and solved multiple scenarios of the problems.
- I have attached all the files in the shared repo, it contains **REPORT.md** which contains the project report.
- I have added **README.md** that contains steps to setup and run the project.
- **Topics** covered ahead:
  - Data Preparation
  - Training
  - Metrics
  - Distributed System Perspective Analysis
  - Raw Outputs

# Data Preparation

- Used dataset from a file on each node based on row number where **row % rank_of_the_process == 0**
- **Synthetic Data** is **polynomial** in nature with some white gaussian noise with a constant mean and standard deviation. Data has been prepared in a way so that it can illustrate the benefits of federated learning even on very short samples and tiny model.
- I have used entire data as **test data** on each process to maintain consistency in evaluation.
- I have plotted the **data distribution** and put it in images folder for your reference.
- **Privacy Preserving Feature:** Since each node has it's own training data, data is not shared between the systems which preserves privacy. Here I have simulated this using the a file reading based on row ids, but it illustrates the privacy idea.

# Training

- In order to **benchmark** the performance, I implemented the same architecture (that I used in federated learning) neural network (NN) with any distributed federated learning to get a **baseline**.
- Developed multiple **variants** of FedAvg method by changing the point of aggregation in various experiments. The **global aggregation** turned out to be best on this specific data and model architecture, which is might not be true in other cases.
- Also, implemented the FedSGD technique where we were doing the **aggregations** at every epoch of the training. Also, tried variation with **synchronisation** after every 10 epochs, which was sub-optimal.
- Trained the models in a way that the results are **reproducible**.

# Metrics

- For the given data and model, **FedSGD** was **closer** to the baseline than FedAvg technique. It’s not conclusive to say FedSGD is better than FedAvg as it **depends on the usecase, model, data and scenario.**
- For **Baseline** Model:
  - Final loss: 1141.0064697265625
  - R-squared: 0.7502739980476314
  - Mean Absolute Error: 27.496908950417136
- For **FedAvg**:
  - After Reducing at Rank 0, Final loss: 1318.7928466796875
  - After Reducing at Rank 0, R-squared: 0.7113628476964176
  - After Reducing at Rank 0, Mean Absolute Error: 28.933150488261308
- For **FedSGD**:
  - After Reducing at Rank 2, Final loss: 1104.7113037109375
  - After Reducing at Rank 2, R-squared: 0.7582177084580953
  - After Reducing at Rank 2, Mean Absolute Error: 27.069782214178403

# Distributed System Perspective Analysis

- **Message Size**:
  - From total number of messages/parameters perspective **per synchronisation**, the number of messages remain same, but since **FedSGD synchronises more frequently** than FedAvg, overall it sends more number of messages in total.
  - Moreover, in practice, since storing **gradients require higher floating precision points** than parameters, FedSGD has even more requirements in terms of overall **message size** capacity than FedAvg.
- **Communication**:
  - **FedSGD** requires **more bandwidth** than FedAvg as it synchronises after every gradient computation while FedAvg runs after a few epochs.
- **Deadlock**:
  - I have used **Reduceall** API of MPI which is a **blocking** API and since there is **no interdependency** anywhere else, deadlock will not occur in this scenario.

# Raw Outputs

*Refer to the DEMO and the REPORT.md*

| Aspect | FedSGD | FedAvg |
| --- | --- | --- |
| Aggregation Method | Aggregates gradients | Aggregates model parameters |
| Update Strategy | Updates global model directly with gradients | Averages model parameters to update global model |
| Communication Overhead | Higher due to gradient transmission | Lower due to parameter transmission |
| Privacy Preservation | Gradients may contain more information about local data | Averaged parameters may preserve privacy better |
| Frequency of Synchronization | Every iteration | After several local updates |

# 2nd Presentation Comments

## Add these to project code and share the updated code today

- Plots
  - Error vs epochs for all the algos
  - Message size plot for fed algo [cumulative messages vs epochs]
  - Number of message [cumulative #msg vs epochs]

# 2nd Presentation Feedback for Future Works

- Implement the algorithms for more realistic scenarios:
  - Async
  - Non-iid
  - Implement on actual distributed systems instead of simulation

- Distributed GPU training
  - NCCL (Nvidia’s Collective Commons Library on top of MPI)
  - CUDA

- Connect with Manaswi ma’am for the above

# Thank you!

*Q&A*