CSCI 635: Introduction to Machine Learning

Assignment 3 - Report

By David Dobbins, Samuel Roberts,
Kirubhakaran Meenakshi Sundaram, Sahil Dayal
11/15/2025

Federated Learning Principles:

Federated Learning is a distributed machine learning paradigm, where the main concept is that data remains decentralized, and the models are trained collaboratively. Instead of searching for databases on the Internet and scrapping through potentially sensitive information, the models are trained locally on an eligible user's data. This allows organizations to leverage large, diverse datasets without ever collecting or storing the raw information centrally. The local model updates are then securely aggregated on a central server to create a shared global model, enabling improved performance while maintaining user privacy. By keeping data on-device and only transmitting learned parameters, Federated Learning reduces privacy risks, minimizes communication overhead, and allows continuous learning across millions of devices in real-world environments.

References:

FL Patterns

Source 1: https://arxiv.org/html/2502.05273v1#S4

Source 2: https://arxiv.org/pdf/2101.02373

Federated Learning Architecture Patterns:

The architecture of federated learning systems can be broadly classified based on three important patterns namely,

1. Coordination pattern: how nodes communicate and coordinate
2. Data Distribution pattern: how data is partitioned across participants
3. Aggregation pattern: how model updates are combined and propagated

1. Coordination Pattern (Communication):
    1. Centralized (Star Topology)
    2. De-centralized (Peer-to-Peer)

3. Hierarchical (Multi-tier)
4. Hybrid

2. Data Distribution Pattern (Data Ownership & Partitioning)
    1. Horizontal FL (Sample-based)
    2. Vertical FL (Feature-based)
    3. Federated Transfer Learning (FTL)
    4. Cross-Silo vs Cross Device

3. Aggregation Pattern (Model Fusion & Update Strategy)
    1. Synchronous Aggregation
    2. Asynchronous Aggregation
    3. Weighted Aggregation
    4. Secure Aggregation
    5. Hierarchical Aggregation

1. Coordination pattern (Communication Topology):

This pattern describes who coordinates training and how updates are exchanged among clients

Pattern	Characteristics	Benefits	Challenges	Use Case/Application	Architecture Examples
Centralized (Star Topology)	A single, trusted central server orchestrates all training rounds, aggregates updates, and broadcasts the global model.	Simple to manage. Faster convergence. Easy global model consensus.	Single point of failure. Bottleneck at the server. Poor scalability for massive networks.	Mobile phone applications (Google Gboard, G-Pay predictions)	FedAvg, FedProx
Decentralized (Peer-to-Per)	Clients exchange model updates directly with their neighboring peers with a central coordinator.	High fault tolerance (no single point of failure). High privacy preservation.	Difficult synchronization. Slower convergence. Inconsistent global model quality.	Consortiums of organizations, IoT networks, Blockchain FL (e.g., Swarm Learning)	D-PSGD,Swarm Learning
Hierarchical (Multi-tier)	Introduces intermediate edge aggregators that group nearby clients,	Reduces communication load on the core network. Leverages	Increased system complexity. Synchronization between hierarchy levels.	Smart cities, 5G networks, geographically distributed sensor networks.	HierFAVG, FedCS

	performing local aggregation before communicating with a central server.	compute at the edge. Better scalability.			
Hybrid	Combines elements, such as multiple regional central servers that sync with a global master server or a decentralized local P2P network coordinated by a central authority.	Optimizes for both local efficiency and global consensus.	High implementation complexity. Requires robust synchronization mechanisms.	Global finance or retail instructions with regional hubs	FedHybrid

2. Data Distribution Pattern (Data Ownership & Partitioning)

This pattern describes how the data is partitioned among the participants, which influences the choice of learning algorithm.

Pattern	Description	Benefits	Challenges	Use Case/Application	Architecture Examples
Horizontal FL (HFL)	Clients share the same feature space but have different samples (rows of data). Data is partitioned horizontally.	Most common FL scenario. Directly leverages standard algorithms like FedAvgs.	Data heterogeneity (Non-IID data) can degrade model performance.	Multiple hospitals in different regions train on the same Electronic Health Record (EHR) schema.	FedAvg, FedProx
Vertical Fl (FHL)	Clients share the same sample space (e.g., same users) but possess different feature spaces (columns of data). Requires secure joining of features.	Allows training across organizations with complementary data. Maintains feature privacy.	Requires secure entity alignment. Complex data joining and feature fusion protocols.	A bank and an e-commerce company collaborating on customer credit scoring.	Split Learning, FATE (WeBank)
Federated Transfer Learning (FTL)	Both sample and feature spaces differ significantly. Knowledge from one domain is transferred to another.	Enables collaboration between diverse domains where HFL/VFL are not possible.	High complexity in designing transfer mechanisms. Lower performance gain that direct FL.	Transferring pre-trained knowledge from public image data to a private, rare disease image dataset.	FTL-based methods
Cross-Silo vs	Cross-Silo:	High data quality	Compliance/legal	Silo: Financial	Secure

Corss-Device	Few (2-100), stable organizations with large, high quality datasets. Cross-Device: Millions of intermittent mobile/IoT devices with small, often, Non-IID, datasets.	(Silo); Immense scale/ diversity (Device).	(silo). High churn/ low bandwidth/ non-IID (Device).	institutions, Pharmaceutical research. Device: Keyboard prediction, Voice assistants.	Aggregation, FedAvg

3. Aggregation Pattern (Model Fusion & Update Strategy)

This pattern focuses on how client updates are collected, combined and synchronized to form the improved global model.

Pattern	Description	Benefits	Challenges	Architecture Examples
Synchronous	The server waits for updates from all (or a minimum subset of) selected clients before performing the aggregation step.	Guarantees consistency and high global model quality. Deterministic training path.	Straggler problem: Slowest client dictates the pace. Poor resource utilization	FedAvg, FedProx
Asynchronous	The server updates the global model immediately upon receiving an update from any client, without waiting for others.	Faster training rounds. Highly robust to stragglers and client dropouts.	Staleness: Updates based on an outdated global model can degrade performance/ stability.	FedAsync, FedBuff
Weighted Aggregation	Client model weights are scaled based on a metric, typically the local dataset size	Ensures clients with more data have a proportionally larger influence on the global model.	May introduce bias towards clients with disproportionately large or poor-quality datasets.	FedAvg, FedNova
Secure Aggregation	Use cryptographic techniques (e.g., Homomorphic Encryption, Secure Multi-Party Computation) to aggregate updates without the server learning individual client updates.	Maximum privacy guarantee against the central server.	Significant computation and communication overhead.	SecureAgg (Bonawitz et al.)

Benefits:

- **Privacy Preservation:** The core benefit is keeping raw data localized, only sharing model updates or encrypted gradients.
- **Access to Diverse Data:** Enables training on vast, real-world data distributed across millions of devices, leading to more robust models.
- **Reduction Communication Cost:** By only sharing smaller model updates (gradients) instead of large raw data, network bandwidth is conserved.
- **Decentralized Control:** Suitable for highly regulated industries (like healthcare and finance) where data cannot leave an organization's premises (cross-silo FL).

Major Challenges:

- **Data Heterogeneity (Non-IID)**: Client data distributions are often Non-Independent and Identically Distributed (Non-IID), which can cause client models to diverge and the global model to perform poorly.
- **System Heterogeneity:** Clients vary greatly in computing power, network bandwidth and battery life, leading to the straggler problem and inconsistent participation.
- **Security & Privacy Attacks:** Model updates can still be used for attacks like Model Inversion (reconstructing training data) or Model Poisoning (injecting malicious data/updates).
- **Communication Efficiency:** Despite sending less data, the sheer number of communication rounds can still be a bottleneck, especially in cross-device settings.

Notable Algorithms:

Algorithm	Description	Coordination	Data Pattern	Aggregation
FedAvg (Federated Averaging)	The foundational FL algorithm. Clients train locally using SGD for E epochs and then send weights/gradients to the server, which averages the models.	Centralized	Horizontal	Synchronous (Weighted)
FedProx	Modifies FedAvg by adding a proximal term to the	Centralized	Horizontal	Synchronous

	client's local loss function. This term penalizes the local model for deviating too much from the previous global model.			
D-PSGD (Decentralized Parallel SGD)	Clients perform Stochastic Gradient Descent (SCG) and only exchange updates with their immediate neighbors in a P2P network.	Centralized	Horizontal	Peer-to-Peer (P2P)
FedAsync	An asynchronous aggregation strategy where the server immediately updates the global model with the received update and then sends the new model back to the client.	Centralized	Horizontal	Asynchronous
SecureAgg (Secure Aggregation)	A cryptographic protocol where the central server can only compute the aggregate sum of all client updates, but cannot read any individual client's update.	Centralized	Cross-Silo	Secure

Heart Failure FedAvg Implementation

Preprocessing: The dataset used was taken from the UC Irvine ML Repository (https://archive.ics.uci.edu/dataset/519/heart+failure+clinical+records) which contains 299 patients who had heart failure with 13 clinical features, the goal being to predict if a patient died during the follow-up period. For preprocessing, the clinical features were separated into the input matrix X and the DEATH_EVENT column into the target vector y. The dataset was split into training and testing sets using an 80/20 split with **stratified sampling** to preserve the class distribution.

Normalization was applied using StandardScaler, with the scaler **fit only on the training data** and then applied to the test data to prevent information leakage.

Federated learning was simulated by dividing **only the training data** into 10 equal client subsets. Each client receives an exclusive shard of the training samples, while the global test set remains centralized for evaluation.

FedAvg Implementation:

We followed the main workflow of the **Federated Averaging (FedAvg)** algorithm for our implementation, but altered it to a simulated environment because we did not have access to genuine distributed devices or medical silos. Instead, we considered the dataset's many partitions as separate *federated clients*, each serving as its own private data holder with no interaction or visibility into other client datasets. This configuration is consistent with typical research-style FL experimentation where realistic evaluations of data heterogeneity, privacy restrictions, and communication constraints are still possible.

From a technical standpoint, the approach begins with prepping the heart failure dataset and splitting it into **ten different client datasets** using numpy.array_split(), as seen in our data prep module. Each client receives a distinct fraction of the normalized training data, while **the global test set remains central**, as evaluation must be consistent across all rounds. This adheres to the typical protocol found in the majority of FL publications, in which **clients only contribute to model updates**; the server is still in charge of **validation and testing**.

Using TensorFlow, we created a **lightweight neural network** with a sigmoid activation and a single dense output layer in the style of **logistic regression**. Instead of creating a deep or over-fitting model, the objective was to keep to a **straightforward, comprehensible model** that accurately depicts FedAvg's behavior when the model is small and communication-friendly (which is the whole point of FL). Our goal was to investigate training stability, performance trends, and the effects of several rounds of averaging rather than pushing a complex design.

Each communication cycle starts with the server initializing a **global model** and choosing a subset of clients at random to take part. This is consistent with FedAvg's default assumption that clients may be unpredictably accessible or unavailable. After receiving the server model weights, each chosen client uses its own subset for **one epoch of local training before returning only the learnt weights; not its data**. After all selected clients have finished, the server uses the

FedAvg algorithm to create the new global model by averaging the weights of each layer element by element. The subsequent communication round is then based on this averaged model.

Without ever having access to raw data, the global model fairly combines knowledge from all clients by repeating this procedure over a number of rounds (we utilized a total of **12 rounds**). Despite the short size of the dataset, we saw that the global model progressively converged with each cycle and that **federated updates were steady rather than bouncing**.

Performance Metrics + Evaluation:

Since this is a **binary classification problem** (patient survives vs. death event), we did not want to depend solely on **accuracy**, which can be misleading, especially when datasets exhibit small imbalances. Predicting the majority class accurately, for instance, can inflate accuracy while failing clinically. We additionally monitored **AUC (Area Under ROC Curve)** and **F1-Score**, which aid in evaluating performance from both a threshold-free and class-balanced standpoint, due to the strong medical significance of this dataset.

Following **each round**, the evaluation was conducted **centrally** using the test set, which was never disclosed to clients. Following each training cycle, we calculated three metrics based on the revised global model's predictions of probability on the centralized test data:

- **AUC** – Measures ranking ability and how well the model separates positive vs. negative cases regardless of threshold.
- **Accuracy** – Specifies how often predictions match the true class label.
- **F1-Score** – Balances precision and recall which is important for medical risk predictions where **false negatives** can be costly (misclassifying a high-risk patient).

The model continuously improved throughout the course of the 12 rounds as the FedAvg updates were implemented. Instead of seeing abrupt fluctuations, the performance steadied, indicating that the architecture and learning rate selected were suitable for the size and variability of this dataset.

The global model consistently obtained high performance metrics (AUC, accuracy, and F1) over rounds, based on the results printed during training. This confirms that the FedAvg algorithm can develop a robust prediction model without requiring the centralization of raw patient data, demonstrating its usefulness in privacy-sensitive healthcare situations, even with **modest local datasets per client** and a straightforward **single-layer design**.

More broadly, our **federated approach demonstrated competitive performance without signal loss from decentralized training**, which is encouraging for potential future experiments involving **noise injection, non-IID distributions, or communication-efficiency differences**.

Sources:

Lo, S., et al. (2021). *Architectural Patterns for the Design of Federated Learning Systems.* **(Focuses explicitly on the Coordination and Aggregation patterns in systems)**
Source: arXiv:2101.02373.

Lian, X., Zhang, C., Zhang, H., Hsieh, C. J., & Liu, J. (2017). *Can Decentralized Algorithms Outperform Centralized Algorithms?* **(Focuses on the Decentralized Coordination Pattern: D-PSGD)**
Source: NeurIPS 2017.

Kairouz, P., et al. (2021). *Advances and Open Problems in Federated Learning.* **(Comprehensive survey covering architectures, non-IID, and challenges)**

- *Source:* Foundations and Trends in Machine Learning, arXiv:1912.04977.