An Overview of Federated Learning

@Jinlin University

Qingfeng Liu Hosei University

August 14, 2025

Federated Learning: Example 1 - Smart Keyboards

- **Application:**
  - Next-word prediction in mobile keyboards (e.g., GBoard)
- **Implementation:**
  - Models learn from typing patterns on-device
  - Only aggregated updates are shared
- **Privacy Benefit:**
  - Personal typing data never leaves the device

Federated Learning: Example 2 - Healthcare

- **Application:**
  - Collaborative disease detection across hospitals
- **Implementation:**
  - Each hospital trains on local patient data
  - Shares only model parameters (not raw data)
- **Regulatory Advantage:**
  - Complies with privacy protection laws — PIPL(CN) / APPI(JP) / HIPAA(US) / GDPR(EU) — by design

Federated Learning: Example 3 - Smart Manufacturing

- **Application:**
  - Predictive maintenance in factories
- **Implementation:**
  - Each machine learns from its own sensor data
  - Combines knowledge without sharing operational data
- **Business Value:**
  - Protects proprietary manufacturing data

Federated Learning: Application Examples

Smart Keyboards

- **Use Case**:
  - Next-word prediction
- **Process**:
  - On-device learning
  - Update aggregation
- **Benefit**:
  - Keeps typing data private

Healthcare

- **Use Case**:
  - Cross-hospital diagnosis
- **Process**:
  - Local model training
  - Parameter sharing
- **Benefit**:
  - PIPL(CN) APPI(JP) HIPAA(US) GDPR(EU) compliant

Smart Manufacturing

- **Use Case**:
  - Predictive maintenance
- **Process**:
  - Sensor data learning
  - Knowledge fusion
- **Benefit**:
  - Protects manufacturing data

What is Federated Learning?

- **Federated Learning (FL)** is a distributed machine learning approach where multiple clients collaboratively train a shared global model without sharing their raw data.
- **Key Idea:**
  - Clients (e.g., smartphones, IoT devices) keep their data locally.
  - Only model updates (gradients or weights) are sent to a central server.
  - The server aggregates updates and sends back the global model.
- Initially proposed by Google in 2016 for on-device learning.

Centralized vs. Federated Learning

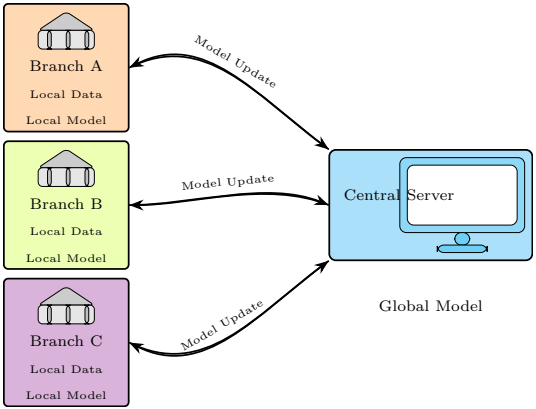
Centralized Learning:

- Data from all clients is uploaded to a central server.
- Global model is trained on aggregated data.
- **Drawbacks:** Privacy concerns, high bandwidth usage.

Federated Learning:

- Data remains on devices.
- Only model parameters are shared.
- **Benefits:** Privacy-preserving, lower bandwidth needs.

Federated Learning Architecture



Components:

- **Central Server:** Coordinates training, aggregates updates.
- **Clients:** Devices with local datasets that train the model.

Workflow:

1. Server sends global model to clients.
2. Clients train model locally on their data.
3. Clients send updated parameters to server.
4. Server aggregates and updates the global model.

Basic Algorithm: Federated Averaging (FedAvg)

- **FedAvg Steps:**
  1. Server initializes the global model w_0.
  2. In each round:
     1. Server selects a subset of clients.
     2. Clients download w_t and perform local training:

$$w_{t+1}^k = w_t - \eta \nabla F_k(w_t)$$

        where F represents the **local loss function** for each client.
     3. Clients upload updated models to server.
     4. Server aggregates:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$$

- **Key:** Weighted averaging based on client data sizes n_k.

Advantages of Federated Learning

- **Privacy:** Raw data never leaves local devices.
- **Bandwidth Efficiency:** Only model updates are sent.
- **Scalability:** Supports millions of clients.
- **On-device Personalization:** Models can adapt to client-specific data.

Challenges in Federated Learning

- **Statistical Heterogeneity:** Clients have Non-IID data.
- **System Heterogeneity:** Clients differ in computation, storage, and network.
- **Communication Overhead:** Frequent exchange of model parameters.
- **Privacy Risks:** Even updates may leak information.

Part II: Overview of Heterogeneous Federated Learning and Challenges

@Jinlin University

Qingfeng Liu Hosei University

August 14, 2025

Overview of HFL and Challenges

Heterogeneous Federated Learning (HFL) is a subfield of FL attracting attention due to its potential applications in large-scale industries.

- **Statistical Heterogeneity**
- **Model Heterogeneity**
- **Communication Heterogeneity**
- **Device Heterogeneity**
- **Additional Challenges**

FL is effective for utilizing rich private data on edge devices like smartphones and wearables without compromising privacy. It enables collaborative ML model training while keeping data decentralized.

Statistical Heterogeneity (Part 1)

One major challenge in HFL is **statistical heterogeneity**, referring to inconsistent data distributions across clients (Non-IID).
Statistical heterogeneity can be divided into four distortion patterns:

1. **Label Skew:** Different label distributions across clients.
   - Label Distribution Skew: Each client holds different types of labels (e.g., handwritten digit recognition where different users have different digits).
   - Label Preference Skew: The same features are annotated differently by different clients due to personal preferences.
2. **Feature Skew:** Different feature distributions across clients.
   - Feature Distribution Skew: Labels consistent, but features vary.
   - Feature Conditional Skew: No overlap of data features across clients.

Statistical Heterogeneity (Part 2) and Model Heterogeneity

1. **Quality Skew:** Annotation or data quality varies across clients.
   - Label Noise Skew: Varying proportions of noisy labels per client.
   - Sample Noise Skew: Clients hold data of different quality levels.
2. **Quantity Skew:** Imbalanced local data sizes among clients.

Model Heterogeneity

Model Heterogeneity: Unlike typical FL, clients may design their own model architectures due to hardware constraints or requirements. Key challenge: transferring knowledge in a model-agnostic way.

- **Partial Heterogeneity:** Some clients share model structures, others differ.

- **Complete Heterogeneity:** All clients use entirely distinct models.

Communication, Device, and Additional Challenges

Communication Heterogeneity: IoT devices run under diverse networks (3G, 4G, Wi-Fi), leading to increased communication costs and reduced learning efficiency.
Device Heterogeneity: Hardware differences (CPU, memory, battery) create system imbalance and inefficiency.
Additional Challenges:

- Knowledge Transfer Barriers: Heterogeneity hinders efficient knowledge sharing.
- Privacy Leakage: Heterogeneity worsens risks of data leakage.

Data-Level Methods (Part 1)

HFL solutions are categorized into three levels:

- **Data-level:** Smooth data heterogeneity, improve privacy.
- **Model-level:** Adapt local models for heterogeneity.
- **Server-level:** Server-based optimizations.

Private Data Processing:

- Data Preparation (e.g., FedMix, Astraea, FAug)
- Quality and diversity enhancement for FL performance.

FedMix: Data Augmentation for FL

- **Goal:** Mitigate Non-IID effects by mixing data distributions across clients.
- **Method:**
  - Uses **mixup** to generate virtual samples:

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \quad \tilde{y} = \lambda y_i + (1-\lambda)y_j$$

    where $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$.

  - Clients exchange a small fraction of their local data to create these mixtures.
- **Effect:** Smooth decision boundaries and reduce client drift.

Astraea: Synthetic Data Generation (Overview)

- **Goal:**
  - To address **label distribution skew** in federated learning (FL).
  - Correct imbalances where some clients lack samples for certain classes.
- **Key Idea:**
  - A global server trains a **generative model** (e.g., GAN or VAE) to produce synthetic data.
  - The server distributes synthetic samples $(\hat{x}, \hat{y})$ to clients to augment their local datasets.
- **Effect:**
  - Reduces data imbalance across clients.
  - Improves fairness and stability of the global model.

Step 1: Training the Global Generator

- The server learns a global data distribution using a generator G_θ.
- The generator takes a latent variable z and produces pseudo-data samples:

$$\min_{\theta} \mathbb{E}_{z \sim p(z)} \left[\mathcal{L} \left(G_\theta(z) \right) \right]$$

- G_θ: generator with parameters θ. - $z \sim p(z)$: latent vector sampled from a prior distribution (e.g., Gaussian). - $\mathcal{L}$: loss function to evaluate quality of generated data.

- Training is done on public or shared datasets that do not compromise privacy.

Step 2 & 3: Generating and Distributing Synthetic Data

- **Step 2: Synthetic Data Generation**
  - The server generates new data using the trained generator:

$$\hat{x} = G_\theta(z), \quad z \sim p(z)$$

  - Labels are assigned using a pre-trained classifier:

$$\hat{y} = f_\phi(\hat{x})$$

    where f_ϕ is the classifier.
- **Step 3: Distributing to Clients**
  - The server sends $(\hat{x}, \hat{y})$ to client k for dataset augmentation:

$$D_k \leftarrow D_k \cup \{(\hat{x}, \hat{y})\}$$

  - Clients use the augmented dataset for local training in the next FL round.

Generative Adversarial Networks (GANs)

- **Goal:** Learn the true data distribution $p_{data}(x)$ and generate new samples that resemble real data.
- **Key Components:**
  - **Generator** $G(z)$**:** Transforms latent variables $z \sim p_z(z)$ into data samples $\hat{x} = G(z)$.
  - **Discriminator** $D(x)$**:** Outputs the probability that input x comes from $p_{data}(x)$ rather than $G(z)$.
- **Minimax Objective:**

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- $D(x)$: maximizes the ability to distinguish real and fake samples. - $G(z)$: minimizes the chance of $D(x)$ detecting $G(z)$ as fake.

- **Learning Process:**
  1. Discriminator D improves to classify real vs. generated data.
  2. Generator G learns to fool D by producing realistic data.
  3. Alternating updates drive $G(z) \rightarrow p_{data}(x)$.

Variational Autoencoders (VAEs): Overview

- **Goal:** Learn a probabilistic generative model to represent data x in a latent space z.
- **Key Components:**
  - **Encoder** $q_\phi(z|x)$**:** Maps data x to a latent distribution over z.
  - **Decoder** $p_\theta(x|z)$**:** Reconstructs data x from latent variable z.
- **Applications:**
  - Data generation
  - Dimensionality reduction
  - Anomaly detection

VAE Data Generation Process

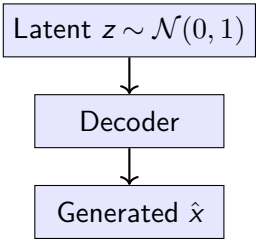
Latent Space to Data Space Mapping

1. Training Phase

- Encoder learns $q(z|x)$, outputs μ, σ
- Reparameterization: $z = \mu + \sigma \cdot \epsilon$
- Decoder learns $p(x|z)$
- Optimizes ELBO objective

2. Generation Phase

- Sample $z \sim \mathcal{N}(0, 1)$
- Generate via $\hat{x} = \text{Decoder}(z)$



Deterministic mapping: latent $\rightarrow$ data space

Variational Autoencoders: ELBO Objective

- **Problem:** Computing $p(x)$ directly is intractable:

$$p(x) = \int p(x|z)\, p(z)\, dz$$

- **Solution: Maximize the Evidence Lower Bound (ELBO):**

$$\log p(x) \geq \mathcal{L}(\theta, \phi; x)$$

$$\mathcal{L} = \underbrace{\mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)]}_{\textbf{Reconstruction loss}} - \underbrace{D_{KL}\big[q_\phi(z|x) \,\|\, p(z)\big]}_{\textbf{Regularization (KL divergence)}}$$

- **Goal:** Maximize $\mathcal{L}$ to make latent representations meaningful and generate realistic samples.

Reparameterization Trick in VAEs

- **Problem:** Direct sampling $z \sim q_\phi(z|x)$ breaks gradient flow during backpropagation.
- **Solution: Reparameterization Trick**
  - Replace stochastic sampling with a differentiable transformation:

$$z = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

  - Here:
    - $\mu_\phi(x), \sigma_\phi(x)$: Encoder network outputs (mean and std dev).
    - ϵ: Random noise, independent of ϕ.
- **Effect:**
  - Enables gradient-based optimization over stochastic latent variables.

FAug: External Data Augmentation

- **Goal:** Improve model robustness in data-scarce environments.
- **Method:**
  - Clients upload **feature embeddings** instead of raw data.
  - Server uses public datasets D_p and clients' embeddings to train augmentation models:

$$\min_{\theta} \sum_{(x_p, y_p)} \left[\underbrace{\mathcal{L}_{\text{task}}\left(f_\theta^{\text{task}}(x_p), y_p\right)}_{\text{Task head}} + \lambda \underbrace{\mathcal{L}_{\text{embed}}\left(f_\theta^{\text{embed}}(x_p), \text{Emb}_k\right)}_{\text{Embedding head}} \right] \tag{1}$$

  - Augmented data is sent back to clients for local training.
  - **Effect:** Enhances local dataset diversity without privacy leakage.

Data-Level Methods (Part 2)

Data Privacy Protection:

- Homomorphic Encryption
- Differential Privacy (DP)
- Data Anonymization

External Data Utilization:

- Knowledge Distillation: FedMD, FedGKT
- Unsupervised Representation Learning: MOON, FedProc

Data Anonymization: Concept

- **What is Data Anonymization?**
  - The process of removing or obfuscating identifiable information from datasets.
- **Why use it in FL?**
  - Ensures datasets cannot be linked back to specific individuals.
  - Protects against deanonymization attacks on shared models.
- **Example:**
  - Masking names, addresses, or unique identifiers before training.

Data Anonymization: Details

- **Techniques:**
  - **K-Anonymity:** Every record is indistinguishable from $k-1$ others.
  - **L-Diversity:** Sensitive attributes are sufficiently diverse in each group.
- **Applications in FL:**
  - Preprocessing client datasets to remove PII before local training.
- **Limitations:**
  - Over-anonymization can degrade data utility.
  - Vulnerable to linkage attacks if auxiliary data is available.

Homomorphic Encryption (HE): Concept

- **What is HE?**
  - A cryptographic technique that allows computations on encrypted data.
  - Ensures the server cannot access raw client data.
- **Why use it in FL?**
  - Model updates may leak private information.
  - HE enables secure aggregation without decrypting updates.
- **Example:**
  - Client encrypts local update $E(w_i)$.
  - Server computes $E(\sum w_i)$ directly without accessing w_i.

Homomorphic Encryption (HE): Details

- **Mathematical Property:**

$$E(m_1) \circledast E(m_2) = E(m_1 \odot m_2)$$

- $\circledast$: operation in ciphertext space - $\odot$: corresponding plaintext operation

- **Applications in FL:**
  - Secure aggregation of encrypted gradients.
  - Server never sees plaintext model parameters.
- **Challenges:**
  - High computational overhead.
  - Large ciphertext size increases communication cost.

Homomorphic Encryption (HE): Details

- **Key Idea:** Perform computations on encrypted data without decrypting it.
- **Mathematical Property:**

$$E(m_1) \circledast E(m_2) = E(m_1 \odot m_2)$$

  - $\circledast$: Operation on ciphertexts (e.g., multiplication)
  - $\odot$: Corresponding plaintext operation (e.g., addition)

- **Example:**

$$E(5) \cdot E(3) = E(5+3) = E(8)$$

  Server works directly on $E(5)$ and $E(3)$ without seeing $5, 3$.

- **Effect:** Enables secure aggregation in FL but increases computational cost.

Differential Privacy (DP): Concept

- **What is DP?**
  - A framework to ensure that the presence/absence of any single record does not significantly affect model outputs.
- **Why use it in FL?**
  - Even without raw data sharing, updates can leak user-specific information.
  - DP introduces noise to mask individual contributions.
- **Key Idea:**
  - Add carefully calibrated noise to gradients or parameters.

Differential Privacy (DP): Details

- **Formal Definition:**

$$\Pr[\mathcal{M}(D_1) \in S] \leq e^{\epsilon} \cdot \Pr[\mathcal{M}(D_2) \in S] + \delta$$

  where D_1 and D_2 are neighboring datasets.

- **In FL:**

$$\tilde{g} = g + \mathcal{N}(0, \sigma^2)$$

  Adds Gaussian noise to gradient updates.

- **Trade-offs:**
  - **Pros:** Strong theoretical privacy guarantees.
  - **Cons:** Added noise may reduce model accuracy.

Gradient Inversion Attack

- **Problem:** Even without raw data sharing, model updates can leak user-specific information.
- **How it Works:**
  - In Federated Learning, clients send gradients $\nabla \mathcal{L}(w)$ to the server.
  - A malicious server uses these gradients to reconstruct the original data:

$$\textbf{Original data} \rightarrow \nabla \mathcal{L}(w) \rightarrow \textbf{Recovered data}$$

  - **Real-world cases:** - User face images reconstructed from gradients (image classification). - Medical records inferred from model updates.
- An adversary can construct a machine learning model that uses gradients as features to predict sensitive attributes of the client's dataset (e.g., age, gender, health status).
- **Implication:** Sensitive client data may be exposed even if raw data never leaves the device.

Diabetes Dataset Leakage

- **Problem:** Federated Learning clients may still reveal dataset-specific information through gradients.
- **Example:**
  - A client trains on a dataset containing only **diabetes patients**.
  - The local gradients strongly reflect features specific to diabetes (e.g., high glucose patterns).
  - A malicious server analyzing gradients can infer:
    - **Presence of diabetes patients in the client dataset.**
    - Sensitive attributes such as age, gender, or lab test values.
- **Solution:** Apply Differential Privacy to add noise to gradients, preventing dataset-level inferences.

Why Gradients Reflect Age Bias – Setup

- **Model:** Assume a linear model for simplicity:

$$f(x; w) = w_0 + w_{\text{age}} x_{\text{age}} + \sum_{j \neq \text{age}} w_j x_j$$

- **Loss Function:** Mean Squared Error (MSE):

$$\mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} (f(x_i; w) - y_i)^2$$

- **Gradient w.r.t. age weight:**

$$\frac{\partial \mathcal{L}}{\partial w_{\text{age}}} = \frac{2}{N} \sum_{i=1}^{N} (f(x_i; w) - y_i)\, x_{i,\text{age}}$$

Effect of Age-Dominant Datasets on Gradient

- In a client dataset consisting only of elderly users:

$$x_{i,\text{age}} \gg 0 \text{ for all } i$$

- Then each term in the gradient sum:

$$\left(f(x_i; w) - y_i\right) x_{i,\text{age}}$$

  will be scaled up by the large value of $x_{i,\text{age}}$.
- **Result:**

$$\left| \frac{\partial \mathcal{L}}{\partial w_{\text{age}}} \right| \text{ becomes large even for moderate prediction errors.}$$

- **Implication:** The server may infer the prevalence of elderly samples from the magnitude of gradients.

Comparison and Privacy Implications

- **In balanced datasets:**
  - x_{age} values vary across samples.
  - Gradient terms may cancel out due to diverse age values.
- **In age-biased datasets:**
  - x_{age} is consistently large across all samples.
  - Gradient becomes biased and more easily distinguishable.
- **Privacy Risk:**
  - A malicious server could use the gradient magnitude as a signal
  - $\rightarrow$ to infer demographic bias (e.g., elderly-only data)
  - $\rightarrow$ or even build classifiers over gradients (property inference attacks)

Model-Level Approaches (Part 1)

Federated Optimization:

- Achieves personalization of local models under statistical heterogeneity while learning global information.
- **Regularization:** Adds penalty terms to the loss function to prevent overfitting.
  - **FedProx:** Adds a proximal term to FedAvg for stability and faster convergence.
  - **FedCurv:** Adapts EWC to prevent catastrophic forgetting.
  - **FedBN:** Solves feature skew via batch normalization layers.
  - **MOON:** Considers regularization between current and previous local models.
- **Meta-learning:** "Learning to learn" by leveraging prior experience.
  - **MAML:** Applicable to any gradient-based method.
  - **Per-FedAvg:** Personalized variant of FedAvg based on MAML.
- **Multi-task Learning:**
  - **MOCHA:** Addresses high communication cost and fault tolerance.
  - **Ditto:** Scalable federated multi-task learning framework.

FedProx: Introduction

- **What is FedProx?**
  - FedProx (Federated Proximal) is an algorithm designed for federated learning.
  - It extends FedAvg to handle **heterogeneous client data** and **systems**.
- **Key Goal:**
  - Mitigate the effects of **statistical heterogeneity (Non-IID data)**.
  - Ensure stable convergence even when clients train on different data distributions.

Limitations of FedAvg

- **FedAvg:** Averages model updates from clients:

$$w^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1}$$

- **Problem:**
  - In Non-IID settings, local models w_k may diverge significantly.
  - Aggregating these updates can slow down or destabilize training.
  - Clients with **heterogeneous computation or communication capacities** exacerbate the issue.

FedProx Algorithm

- **Key Idea:**
  - Add a **proximal term** to the client's local objective:

$$\min_w f_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

where:

- $f_k(w)$: local loss on client k,
- w^t: global model parameters,
- μ: proximal coefficient.

- **Effect:**
  - Keeps client updates close to the current global model.
  - Reduces drift between client and server models.

Advantages of FedProx

- **Handles System Heterogeneity:**
  - Supports clients with varying computational power and participation levels.
- **Improves Convergence:**
  - Stabilizes federated learning in highly Non-IID scenarios.
- **Flexibility:**
  - FedProx reduces to FedAvg when $\mu = 0$.

FedProx in Real-World Applications

- **Domain:** Healthcare Federated Learning
  - Multiple hospitals collaborate to train a global model for disease prediction (e.g., diabetes, cancer).
  - Patient data remains **private and never leaves each hospital**.
- **Challenge:**
  - Data distributions differ significantly across hospitals (**Non-IID**):
    - Hospital A: Mostly elderly patients
    - Hospital B: Younger population
    - Hospital C: Rare diseases

Problem with FedAvg in Healthcare

- **FedAvg limitation:**
  - Local models may diverge due to highly skewed data distributions.
  - Global model fails to generalize across all hospitals.
- **Example:**
  - Hospital A's model prioritizes elderly patients' features.
  - Hospital B's updates override A's during aggregation.
- **Result:**
  - Unstable training and poor global performance.

FedProx Solution in Healthcare

- **Key Mechanism:**
  - Adds a proximal term to local loss:

$$f_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

  - Encourages hospitals' updates to stay close to the global model.
- **Effect:**
  - Reduces the impact of Non-IID data.
  - Stabilizes training across diverse patient populations.

Impact of FedProx in Healthcare

- **Benefits:**
  - Improves model accuracy across all hospitals.
  - Preserves privacy and complies with regulations (e.g., HIPAA, GDPR).
  - Handles clients with different computational and communication resources.
- **Real-World Deployment:**
  - Federated healthcare systems using FedProx have been explored for:
    - COVID-19 detection from CT scans
    - Diabetes prediction from EHRs (Electronic Health Records)

Model-Level Approaches (Part 2)

Knowledge Transfer across Models:

- **Knowledge Distillation:**
  - **FedMD:** Uses logits from local models for consensus building.
  - **FedGEN:** Data-free knowledge distillation for statistical HFL.
- **Transfer Learning:**
  - **FedHealth:** Federated transfer learning in healthcare.
  - **Def-KT:** Point-to-point knowledge transfer without server participation.
  - **FedPer:** Global base layers trained on server, personalized layers locally.

Architecture Sharing:

- **Backbone Sharing:** FedPer, FedRep
- **Classifier Sharing:** LG-FedAvg
- **Other Part Sharing:** HeteroFL

Server-Level Approaches (Part 1)

Client Selection:

- Optimizes client subsets for participation in FL iterations.
- **Favor:** Experience-driven client selection framework.
- **FedCS:** Considers data resources, computation capacity, and channel conditions.

Client Clustering:

- **FL+HC:** Hierarchical clustering based on similarity of updates.
- **CFL:** Clusters clients by cosine similarity of gradient updates.
- **FLAME:** Detects adversarial updates via clustering.

Server-Level Approaches (Part 2)

Decentralized Communication:

- Removes dependency on a central server for robustness.
- **BrainTorrent:** Peer-to-peer FL framework.
- **Combo:** Segmented gossip approach.
- **ProxyFL:** Clients exchange proxy models for privacy.
- **BFLC:** Blockchain-based decentralized FL framework.

Future Directions (Part 1)

- **Improving Communication Efficiency:**
  - Reduce costs and delays from heterogeneous networks.
  - Balance communication efficiency and model accuracy.
- **Federated Fairness:**
  - Address free-rider issues and bias towards frequent contributors.
- **Strengthening Privacy Protection:**
  - Fine-grained privacy constraints per client and sample.

Future Directions (Part 2)

Attack Robustness:

- **Poisoning Attacks:** Data or model poisoning (e.g., DBA attack).
- **Inference Attacks:** Infers sensitive client data.
- **Defense Strategies:**
  - **CRFL:** Gradient clipping and noise addition.
  - **RBML-DFL:** Blockchain-based robust FL.
  - **TEE:** Trusted Execution Environments for secure computation.

Future Directions (Part 3)

Establishing Uniform Benchmarks:

- Develop widely recognized benchmark datasets and test frameworks.
- **Existing FL Systems:** FedML, FedScale
- **Specialized FL Systems:** FedReIDBench, pFL-Bench
- **Datasets:** LEAF, object detection datasets
- Importance: Promotes reproducibility and fair evaluation of security, convergence, accuracy, and generalization.

Part III: Vertical Federated Learning

@Jinlin University

Qingfeng Liu Hosei University

August 14, 2025

Overview

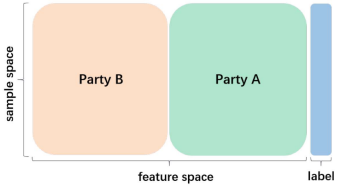
- **Vertical Federated Learning (VFL)** enables multiple organizations, holding different features of the same user group, to collaboratively train machine learning models **without exposing raw data or model parameters**.
- Given the rapid progress in VFL research and applications, this survey provides a comprehensive review of:
  - VFL concepts and algorithms
  - Effectiveness, efficiency, privacy, and fairness
  - Privacy attacks and defense strategies
  - A unified framework "**VFLow**"
  - Industrial applications and open challenges

Three Types of Federated Learning

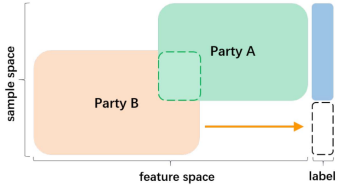
- **Federated Learning (FL)** is a machine learning paradigm where multiple parties collaboratively build models **without centralizing their data**.
- Proposed by Google in 2016 for cross-device scenarios.
- Extended to **cross-silo collaboration**, where trusted organizations join to train models.

Three Types of Federated Learning

(a) Horizontal Federated Learning



(b) Vertical Federated Learning



(c) Federated Transfer Learning

Three Main Categories of FL

- **Horizontal FL (HFL)**: Parties hold data with different users but the same feature space
- **Vertical FL (VFL)**: Parties share the same users (Sample) but have different feature spaces
- **Federated Transfer Learning (FTL)**: Both users and features differ
- **HFL vs. VFL**:
  - HFL shares model updates (e.g., weights, gradients)
  - VFL exchanges intermediate results
  - VFL typically belongs to cross-silo FL

What is Vertical Federated Learning (VFL)?

- **Definition**: VFL enables multiple parties with **different feature spaces** but **shared sample IDs** to jointly train a model without sharing raw data.
- **Scenario**:
  - Party A: $(X_A, y) \rightarrow$ features X_A and labels y
  - Party B: $X_B \rightarrow$ different features X_B for same users
- **Goal**: Learn model parameters collaboratively:

$$\hat{y} = f(W_A X_A + W_B X_B)$$

Basic Architecture of VFL

- **Parties Involved**:
  - **Active Party**: owns labels y
  - **Passive Party**: no labels
- **Training Process**:
  - Each party computes local output:

$$z_A = W_A X_A, \quad z_B = W_B X_B$$

  - Aggregate logits securely:

$$z = z_A + z_B$$

  - Compute loss and gradients collaboratively without revealing raw data.
- **Key Technique**: Secure computation (e.g., encryption, secure aggregation).

Loss Computation and Gradient Update

- **Loss Function** (Active Party):

$$L = \frac{1}{N} \sum_i \ell(f(z_A^i + z_B^i), y^i)$$

- **Gradient Sharing** (Secure):
  - Active party computes gradient:

$$\nabla_{z_A}, \nabla_{z_B}$$

  - Send encrypted gradients to parties to update W_A, W_B:

$$W_A \leftarrow W_A - \eta \nabla_{W_A}, \quad W_B \leftarrow W_B - \eta \nabla_{W_B}$$

Applications and Challenges

- **Applications**:
  - **Finance**: Bank A (transaction data), Bank B (credit ratings)
  - **Healthcare**: Hospital A (genetic data), Hospital B (imaging data)
- **Challenges**:
  - Data privacy (encryption overhead)
  - Communication cost
  - Label availability (only one party has labels)

Horizontal vs. Vertical FL

Aspect	Horizontal FL (HFL)	Vertical FL (VFL)
Data Split	Same features, different users	Same users, different features
Parties' Data	$(X_A, y_A), (X_B, y_B)$: different user IDs	Party A: (X_A, y), Party B: (X_B)
Goal	Train a global model over all users	Combine feature spaces for better predictions
Typical Scenario	Hospitals in different regions	Banks with complementary customer information
Key Challenge	Data heterogeneity (Non-IID)	Feature alignment & secure aggregation

Table: Comparison between Horizontal FL and Vertical FL.

Horizontal vs. Vertical FL

- **HFL Equation**:

$$\hat{y} = f(WX), \quad X = \text{Union of all clients' data}$$

- **VFL Equation**:

$$\hat{y} = f(W_A X_A + W_B X_B)$$

Vertical Federated Learning (VFL) in Detail

- **Definition**: Parties hold different features for the same user samples and collaborate to train a model.
- **Roles of Parties**:
  - **Active Party**: Holds label information
  - **Passive Parties**: Hold only features
- **Why VFL?**
  - Data silo issues in industry (fragmented datasets across organizations)
  - Rising privacy and data security regulations worldwide

VFL Framework – Problem Definition and Model Structure

- **Goal**: Collaboratively train models while preserving data privacy and security
- **Model decomposition**:
  - Local models G_k
  - Global module F_K
- **Loss function**:

$$\min_{\Theta} \mathcal{L}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} f(\Theta; x_i, y_i)$$

VFL Variants and System Architecture

- **splitVFL**: Trainable F_K (e.g., splitNN)
- **aggVFL**: Aggregation-only F_K
- Some variants where active parties hold no features

VFL Training Protocol

1. **Privacy-Preserving Entity Alignment**:
   - Private Set Intersection (PSI) for aligning sample IDs
2. **Privacy-Preserving Training**:
   - Gradient descent on local data
   - Exchange of intermediate outputs H_k and gradients $\partial L/\partial H_k$

Improving Communication Efficiency

- **Key techniques**:
  - Multiple client updates (FedBCD)
  - Asynchronous coordination (GP-AVFL)
  - One-shot communication
  - Compression and feature selection

Improving Effectiveness

- **Self-Supervised Learning (SSL)**: FedHSSL
- **Semi-Supervised Learning (Semi-SL)**: FedCVT
- **Knowledge Distillation (KD)**: VFL-Infer
- **Transfer Learning (TL)**: SFTL

Privacy-Preserving Protocols

- P-1: Basic protocol
- P-2: Encrypt intermediate results (e.g., Homomorphic Encryption)
- P-3: No training info exposed (MPC)
- P-4: Even trained models are protected

Data Inference Attacks and Defenses

- **Label Inference Attacks**: Infer labels from gradients
- **Feature Inference Attacks**: Model inversion, gradient inversion
- **Defenses**:
  - Cryptographic: HE, MPC
  - Non-cryptographic: Noise injection

Backdoor Attacks and Defenses

- **Attack types**:
  - Targeted: LRB, ADI
  - Non-targeted: Adversarial samples
- **Defenses**: CAE, DCAE, RVFR

VFLow Framework and Applications

- **VFLow**: Balances utility, privacy, and efficiency
- **Applications**:
  - Recommendation systems
  - Financial risk management
  - Healthcare (e.g., MIMIC-III dataset)

Conclusions and Future Directions

- **Open challenges**:
  - Lack of interoperability across platforms
  - Explainability and fairness
  - Automated VFL
- **Takeaway**: VFL is a promising solution for cross-organization data silos

Part IV: Incentive Mechanism for FL

@Jinlin University

Qingfeng Liu Hosei University

August 14, 2025

Zhan et al. (2020) Review

- **Title:** A Learning-based Incentive Mechanism for FL
- **Authors:** Yufeng Zhan, Peng Li, et al.
- **Summary:**
  - IoT devices generate massive edge data.
  - **Centralized DL faces bandwidth, storage, privacy challenges.**
  - FL enables edge training without exposing raw data.
  - Proposes a **Deep Reinforcement Learning (DRL)-based incentive mechanism** for FL.

Federated Learning (FL): Definition

- **FL enables collaborative training across distributed nodes.**
- Process:
  1. Server distributes initial model.
  2. Edge nodes train locally.
  3. Nodes send updates, not raw data.
  4. Server aggregates updates into global model.
- **Advantages:**
  - Data privacy preserved.
  - Handles distributed, large-scale datasets.

Challenges for Incentive Mechanisms (1/2)

- **Why incentives?**
  - Edge nodes need motivation to contribute resources.
- **FL-specific difficulties:**
  - **Information un-sharing:**
    - Nodes keep local decisions private (e.g., data amount).
  - **Contribution evaluation:**
    - Non-linear relation between accuracy and data amount.
    - Hard to predict data quality/model impact.

Challenges for Incentive Mechanisms (2/2)

- **Freshness Requirements:**
  - Many IoT apps require fresh data for training.
- **Existing incentive designs fail:**
  - Assumes shared information or predictable contributions.
  - Lacks dynamic adaptability.

Proposed Solution: Game-Theoretic Framework

- **Stackelberg Game:**
  - **Leader:** Parameter server chooses payment τ.
  - **Followers:** Edge nodes choose participation level x_n.
- **Objective:**
  - Server: maximize $u(\tau) = \lambda g(X) - \tau$.
  - Edge nodes: maximize $u_n(x_n) = \frac{x_n \tau}{\sum x_m} - cost$.

Proposed Solution: Game-Theoretic Framework

- **Stackelberg Game:**
  - **Leader:** Parameter server chooses payment τ.
  - **Followers:** Edge nodes choose participation level x_n.
- **Objective:**
  - **Server:**
$$u(\tau) = \lambda g(X) - \tau$$
  - **Edge nodes:**
$$u_n(x_n) = \frac{x_n \tau}{\sum x_m} - cost$$
- **Notation:**
  - τ: Payment offered by the server to incentivize participation.
  - x_n: Participation level of edge node n.
  - $X = \sum x_n$: Total participation from all edge nodes.
  - λ: Utility scaling factor for the server.
  - $g(X)$: Benefit function of aggregated participation.
  - $cost$: Local cost incurred by each edge node.

What is a Stackelberg Game?

- A strategic game involving two roles:
  - **Leader:** Moves first and selects a strategy.
  - **Follower:** Observes the leader's decision and responds optimally.
- **Key Characteristics:**
  - Sequential decision-making: leader acts first, follower reacts.
  - The leader anticipates the follower's best response.
- **Example:**
  - A company (leader) sets a product price.
  - Consumers (followers) decide how much to buy based on the price.
- **Applications in FL:**
  - Server (leader): determines incentives.
  - Clients (followers): choose participation levels in response.

Game Analysis: Full Information

- For fixed τ, unique Nash equilibrium exists.
- Stackelberg equilibrium proven to exist for leader-follower setup.
- Server anticipates node reactions when setting payment.

DRL-based Incentive Mechanism (1/2)

- **Why DRL?**
  - Overcomes incomplete info and dynamic environments.
- **Parameter Server:**
  - Learns optimal payment strategy $\pi(\tau_t|s_t, \theta)$.
- **Edge Nodes:**
  - Learn participation strategies $\pi_n(x_t^n|s_t^n, \theta_n)$.

DRL-based Incentive Mechanism (2/2)

- **Learning Process:**
  1. Server observes state s_t, selects payment τ_t.
  2. Edge nodes react and train models locally.
  3. Server aggregates models, updates policy using PPO.
- **Algorithm:**
  - Actor-Critic framework with PPO optimizer.

PPO and Server Policy Updates (Concept)

- **Server aggregates models:**
  - Clients perform local training and send updated models (w^k) to the server.
  - The server aggregates them to produce a new global model:

$$w^{t+1} = \sum_k \frac{n_k}{n} w^k$$

- **Server updates policy using Proximal Policy Optimization (PPO):**
  - Learns a policy $\pi_\theta(\tau|s)$ to decide payments τ.
  - Optimizes the PPO objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(\tau|s)}{\pi_{\theta_{old}}(\tau|s)}$.

PPO Objective: Intuition

- **Goal:** Improve the policy $\pi_\theta(\tau|s)$ smoothly without large destructive updates.
- **Key Idea:**
  - Avoid "big jumps" in policy updates by restricting changes to stay close to the old policy $\pi_{\theta_{\text{old}}}$.
  - Introduce a probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(\tau|s)}{\pi_{\theta_{\text{old}}}(\tau|s)}$$

  - $r_t(\theta) > 1$: action τ becomes more likely. - $r_t(\theta) < 1$: action τ becomes less likely.
- **Advantage:** $\hat{A}_t$ measures how good the action is at time t.

PPO Objective: Clipping Mechanism

- **Objective function:**

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right]$$

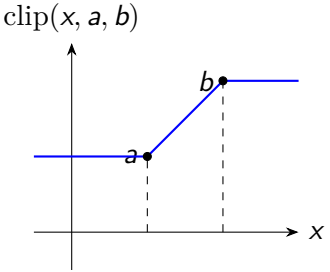
- **Clipping effect:**
  - Prevents $r_t(\theta)$ from moving too far from 1.
  - If $r_t(\theta)$ tries to exceed $(1+\epsilon)$ or drop below $(1-\epsilon)$, the objective is flattened.
- **Why?**
  - Stabilizes training and avoids catastrophic policy updates.

Clip Function in PPO

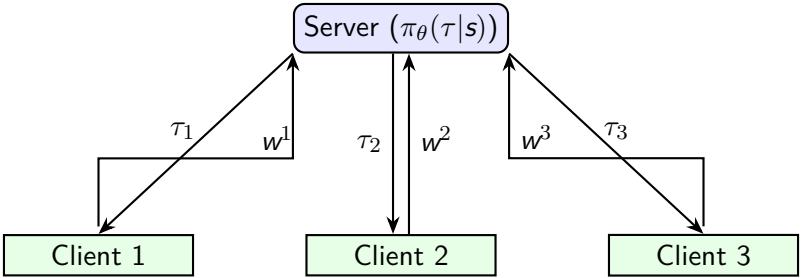
Definition:

$$\text{clip}(x, a, b) = \begin{cases} a & \text{if } x < a, \\ x & \text{if } a \le x \le b, \\ b & \text{if } x > b \end{cases}$$

Here: $a = 1 - \epsilon$, $b = 1 + \epsilon$

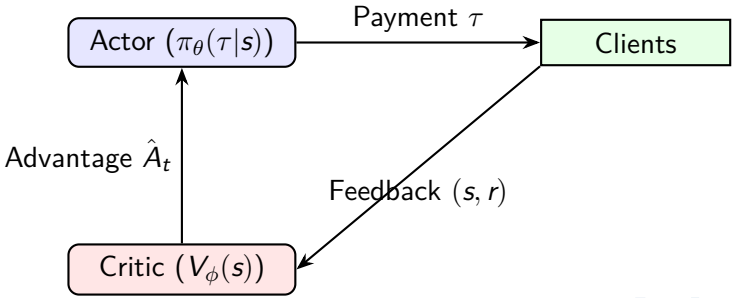


PPO and Server Policy Updates (Diagram)



Actor-Critic Framework in FL

- **Actor:**
  - Chooses payments τ for clients based on state s.
  - Represents the policy $\pi_\theta(\tau|s)$.
- **Critic:**
  - Estimates the value function $V_\phi(s)$ and advantage $\hat{A}_t$.
  - Provides feedback for improving the Actor's policy.



System Model: Utility Functions

- **Edge Node Utility:**

$$u_n(x_n) = \frac{x_n \tau}{\sum x_m} - c^n_{com} x_n - c^n_{cmp} x_n$$

- **Server Utility:**

$$u(\tau) = \lambda g(X) - \tau, \quad X = \sum x_n$$

Performance Evaluation: Convergence

- **Experiment Setup:**
  - Tensorflow 1.9, Ubuntu 16.04
  - $g(X) = 10\ln(1 + X)$
- **Results:**
  - Server payment converges to Stackelberg equilibrium.
  - Edge node strategies converge to Nash equilibrium.

Performance: Training Cost Impact

- Higher node cost $\Rightarrow$ lower participation.
- Server utility decreases with higher costs.
- DRL mechanism outperforms random/greedy baselines.

Performance: Node Count Impact

- More edge nodes $\Rightarrow$ server utility increases.
- Average node utility decreases (competition effect).
- DRL adapts to changing system size.

Conclusion

- Proposed **DRL-based incentive mechanism** for FL.
- Proved existence of equilibria.
- Experiments show superior performance.
- Enables practical, dynamic FL in IoT.

Incentive-Compatible Federated Learning

@Jinlin University

Qingfeng Liu Hosei University

August 14, 2025

Javaherian et al. (2025) Review

- **Title:** Incentive-Compatible Federated Learning with Stackelberg Game Modeling
- **Authors:** Simin Javaherian, Bryce Turney, Li Chen, Nian-Feng Tzeng
- **Published:** January 5, 2025 (arXiv preprint)

Summary:

- Proposes a novel federated learning framework (**FLamma**, FL+γ) based on Stackelberg game theory.
- Addresses fairness and incentivization among heterogeneous clients.
- Introduces a **decay factor (γ)** for balancing client contributions dynamically.
- Derives optimal strategies for the server (leader) and clients (followers) to reach Stackelberg equilibrium.

Current Challenges in FL

- **Heterogeneous environments**:
  - Clients differ in resources and capabilities
  - Can reduce system-wide performance
- **Limitations of existing approaches**:
  - Focus on maximizing global model accuracy
  - Often neglect fairness among clients and system efficiency
  - Assume clients are always motivated to participate
- **Need for Incentives**:
  - FL systems require incentives to attract and retain client participation

Introducing FLamma

- **FLamma**: A new FL framework based on an adaptive gamma-based Stackelberg game
- **Goals**:
  - Address existing shortcomings
  - Promote fairness by modeling client behavior and resource allocation
- **Key Idea**:
  - Server acts as **Leader** and dynamically adjusts decay factor γ
  - Clients act as **Followers** and choose their local epochs τ_i to maximize utility

Stackelberg Game for Incentive Design

- **Game-Theoretic Approach**:
  - Aligns client goals with the network-wide objective
- **Stackelberg Game Concept**:
  - **Leader (Server)** acts first, setting decay factor γ
  - **Followers (Clients)** observe and respond by selecting optimal τ_i
- **Goal**: Achieve Stackelberg Equilibrium for fairness and stability

Utility Design in FLamma

- **Server Utility**:

$$U_{server}(\gamma, \tau_i) = \sum \left[\gamma \cdot \left(1 - \frac{\|w_i^t - w^t\|}{\|w^t\|} + \tau_i \right) \right] - t \cdot \gamma^2$$

- **Client Utility**:

$$U_i(\gamma, \tau_i, \tau_{-i}) = \gamma \cdot \left(1 - \frac{\|w_i^t - w^t\|}{\|w^t\|} \right) \cdot \tau_i - c_i \cdot \tau_i^2$$

- **Individual Rationality (IR)**:

$$U_i(\gamma, \tau_i, \tau_{-i}) \geq 0$$

Optimal Strategies for Server and Clients

- **Optimal Local Epoch for Client** i:

$$\tau_i^* = \frac{\gamma \cdot \left(1 - \frac{\|w_i^t - w^t\|}{\|w^t\|}\right)}{2c_i}$$

- **Optimal Decay Factor for Server**:

$$\gamma^* = \frac{\left(1 - \frac{\|w_i^t - w^t\|}{\|w^t\|}\right) \cdot c_i}{2t \cdot c_i - \left(1 - \frac{\|w_i^t - w^t\|}{\|w^t\|}\right)}$$

- **Existence of Nash Equilibrium**:
  - Subgames have at least one Nash equilibrium

FLamma Algorithm Flow

1. **Server Side**:
   - Calculate client contributions ω_i
   - Select client subset S_t
   - Broadcast (w^t, γ) to selected clients
2. **Client Side**:
   - Choose τ_k to maximize utility
   - Train and send updates to server
3. **Server Aggregates**:
   - Update global model w^{t+1} and decay factor γ

How FLamma Promotes Fairness

- **Decay Factor** γ:
  - Dynamically reduces influence of dominant clients
- **Prevents Overfitting**:
  - Ensures balanced contribution from all clients
- **Result**:
  - Lower accuracy variance among clients
  - Improved system robustness

Convergence Analysis of FLamma

- **Competitive Convergence Rate**
- **Decay Factor** γ:
  - Ensures stability
  - Limits excessive influence of high-local-epoch clients
- **FedAvg Comparison**:
  - FLamma achieves similar behavior with bounded γ

Experimental Setup and Results

- **Datasets**: MNIST, FashionMNIST, CIFAR10
- **Models**: LeNet-5, ResNet-18
- **Baselines**: FedAvg, FedProx, q-FFL, Incentivization
- **Metrics**: Test Accuracy, Accuracy Variance

Results: IID Data

- FLamma outperforms baselines in both accuracy and fairness

- **CIFAR10**: FLamma improves accuracy by 11.59% and reduces variance by 82.6%

Results: Non-IID Data

- FLamma shows significant improvements over baselines in challenging non-IID settings
- **CIFAR10**: 120.93% accuracy improvement
- **FMNIST**: 99.09% variance reduction

Strengths and Limitations

- **Strengths**:
  - Significant fairness improvement
  - Maintains competitive global accuracy
  - Encourages active client participation
- **Limitations**:
  - Precise contribution measurement is challenging
  - Gamma tuning requires experimentation

Conclusions and Future Work

- **FLamma** leverages Stackelberg game modeling for incentive-compatible federated learning
- Promotes fairness by dynamically adjusting client contributions
- **Future Directions**:
  - Automate gamma tuning
  - Integrate reinforcement learning for dynamic optimization