People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة عين تموشنت بلحاج بوشعيب
University of Ain Temouchent - Belhadj Bouchaib
Faculty of Science and Engineering
Department of Mathematics and Computer Science

Projet de Fin d'Etudes
Pour l'obtention du diplôme de Master en : Informatique
Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Cybersécurité et intelligence artificielle

Thème

Sujet: Study of Security and Privacy Challenges in Federated Learning Systems

Présenté par :

- Mr. Younes ilies
- Mr .Zingara mohamed seddik.

Devant le jury composé de

Président	Dr Bouafia Zoheir
Examinateur	Dr MERAD BOUDIA Djalal
Encadreur	Dr Benaribi Fethi Imad
Co-Encadreur	Dr Ouchani Samir

Année Universitaire : 2024/2025

Acknowledgements:

First and foremost, we express our deepest gratitude to Allah for giving us the strength and perseverance to complete this project.

We would like to sincerely thank our supervisor, **Dr. Benaribi Fethi Imad**, for his invaluable guidance, encouragement, and support throughout this work. His expertise and availability were instrumental in the completion of this academic work.

We are equally grateful to our co-supervisor, **Dr. Ouchani Samir**, whose constructive feedback and technical insights greatly contributed to shaping our research.

We extend our appreciation to the **members of the examination committee**, whose careful review and feedback will undoubtedly help us improve the quality of our work.

Our heartfelt thanks also go to all our professors for their dedication and for generously sharing their knowledge throughout our academic journey.

Finally, we thank our families and friends for their unwavering moral and emotional support, which helped us persevere through every stage of this project. To everyone who contributed, directly or indirectly, we offer our sincere gratitude.

We wish to extend our warmest thanks to our parents, siblings, and friends, who provided us with moral and intellectual support throughout our academic journey.

To all those who, from near or far, contributed with their advice, encouragement, and friendship to the development of this modest work, please find here the expression of our deep gratitude.

Rrésumé:

L'apprentissage fédéré (FL) représente une avancée majeure face à la prolifération des données sur les appareils périphériques, permettant l'entraînement collaboratif de modèles d'apprentissage automatique sans centralisation des données brutes. Cette approche préserve la confidentialité des utilisateurs et réduit les coûts de transfert de données, trouvant des applications cruciales dans des domaines sensibles comme la santé et la finance. Cependant, le partage des mises à jour de modèle expose les systèmes FL à de nouvelles vulnérabilités, notamment les attaques par empoisonnement de modèle et les attaques par inférence, qui compromettent la confidentialité et l'intégrité du système.

Cette thèse vise à approfondir la compréhension des principes fondamentaux de l'apprentissage fédéré et à investiguer expérimentalement diverses attaques ciblant tant les aspects de sécurité que de confidentialité des systèmes FL. En matière de sécurité, une attention particulière est portée aux attaques contre les agrégateurs de modèles, tels que FedGA, en analysant leurs vulnérabilités aux manipulations malveillantes qui pourraient nuire à l'intégrité du modèle global. Côté confidentialité, l'étude explore les risques de fuite d'informations sensibles à partir des mises à jour de modèles locaux. Par cette double investigation, la thèse cherche à mettre en lumière les forces et les faiblesses des cadres FL actuels et à contribuer au développement de systèmes d'apprentissage distribués plus robustes et sécurisés.

Mot cle : Apprentissage Fédéré (FL), Sécurité en FL, Confidentialité en FL, FedGA.

Abstract:

Federated Learning (FL) represents a major advancement in the face of data proliferation on edge devices, enabling collaborative training of machine learning models without centralizing raw data. This approach preserves user privacy and reduces data transfer costs, finding crucial applications in sensitive areas such as healthcare and finance. However, sharing model updates exposes FL systems to new vulnerabilities, particularly model poisoning attacks and inference attacks, which compromise system confidentiality and integrity.

This academic work aims to deepen the understanding of the fundamental principles of Federated Learning and to experimentally investigate various attacks targeting both security and privacy aspects of FL systems. In terms of security, particular attention is paid to attacks against model aggregators, such as FedGA, by analyzing their vulnerabilities to malicious manipulations that could harm the integrity of the global model. On the privacy side, the study explores the risks of sensitive information leakage from local model updates. Through this dual investigation, the academic work seeks to highlight both the strengths and weaknesses of current FL frameworks and to contribute to the development of more robust and secure distributed learning systems.

Keywords: Federated Learning (FL), FL Security, FL Privacy, Model Aggregation, FedGA

ملخص:

تقدمًا كبيرًا في مواجهة انتشار البيانات على الأجهزة الطرفية، مما يتيح التدريب التعاوني لنماذج التعلم الآلي دون مركزة البيانات الخام. يحافظ هذا النهج على خصوصية المستخدم ويقلل من تكاليف نقل (FL) يمثل التعلم الموحد البيانات، ويجد تطبيقات حاسمة في مجالات حساسة مثل الرعاية الصحية والتمويل. ومع ذلك، فإن مشاركة تحديثات النموذج تعرض أنظمة التعلم الموحد لنقاط ضعف جديدة، لا سيما هجمات تسميم النموذج وهجمات الاستدلال، والتي تعرض سرية النظام وسلامته للخطر.

تهدف هذه الأطروحة إلى تعميق فهم المبادئ الأساسية للتعلم الموحد وإجراء تحقيق تجريبي في مختلف الهجمات التي تستهدف جوانب الأمان والخصوصية لأنظمة التعلم الموحد. فيما يتعلق بالأمان، يتم إيلاء اهتمام خاص للهجمات ضد من خلال تحليل نقاط ضعفها تجاه التلاعبات الخبيثة التي قد تضر بسلامة النموذج العام. ومن جانب الخصوصية، تستكشف الدراسة مخاطر تسرب المعلومات الحساسة من تحديثات النماذج المحلية. من FedGA مجمعي النماذج، من خلال هذا التحقيق المزدوج تسعى الأطروحة إلى تسليط الضوء على نقاط القوة والضعف في أطر التعلم الموحد الحالية والمساهمة في تطوير أنظمة تعلم موزع أكثر قوة وأمانًا.

الكلمات المفتاحية: التعلم الموحد (FL), FL حماية, FL خصوصية, تجميع النموذج, FedGA.

Contents

list of figures:

list of figures:

list of tables:

List of abbreviations and acronyms:

AI: Artificial Intelligence

Adam: Adaptive Moment Estimation

BGD: Batch Gradient Descent

CNN: Convolutional Neural Networks

CM: Coordinate-Wise Median

CTM: Coordinate-Wise Trimmed Mean

CHAP : Challenge-Handshake Authentication Protocol

DL: Deep Learning

DoS: Denial of Service

DDoS : Distributed Denial of Service

FedAvg : Federated Averaging

FedSGD: Federated Stochastic Gradient Descent

FedProx: Federated Proximal

FedNova: Federated Normalized Averaging

FedGA: Federated Genetic Algorithm

FL: Federated Learning

FTL: Federated Transfer Learning

Fp : False Positives

Fn : False Negatives

GDPR: General Data Protection Regulation

GAN : Generative Adversarial Network

HIPAA: Health Insurance Portability and Accountability Act

HTTPS : Hypertext Transfer Protocol Secure

HTTP : Hypertext Transfer Protocol

ICPS: Intelligent Cyber-Physical Systems

IoT: Internet of Things

LSTM: Long Short-Term Memory Networks

ML: Machine Learning

MBGD: Mini-batch Gradient Descent

MIA :Membership Inference Attack

MitM : Man-in-the-Middle

OTP : One-Time Password

P2P: Peer-to-Peer

PIA :property Inference Attack

RMSProp: Root Mean Square Propagation

RNN: Recurrent Neural Networks

ReLU: Rectified Linear Unit

SGD: Stochastic Gradient Descent

SGD: Stochastic Gradient Descent

SSL : Secure Sockets Layer

Tanh: Hyperbolic Tangent

Tp :True Positives

Tn :True Negatives

TLS : Transport Layer Security

General Introduction:

The rise of edge devices—smartphones, sensors, and wearables—has led to an explosion of decentralized data. This ever-expanding data landscape demands innovative machine learning (ML) solutions that can effectively utilize this information while simultaneously safeguarding user privacy. Federated Learning (FL) emerges as a powerful paradigm to address this challenge. It enables collaborative model training across numerous client devices without requiring the raw data to be centrally aggregated. Unlike conventional ML approaches that depend on centralized data collection, FL decentralizes the learning process, significantly mitigating privacy risks and communication overhead.

FL has gained considerable momentum across diverse domains, including healthcare, finance, and personalized recommendation systems, where data privacy is of utmost importance. However, the inherent distributed nature of FL introduces unique and complex security and privacy vulnerabilities. While raw data remains securely on user devices, the model updates exchanged during training can inadvertently leak sensitive information, making systems susceptible to various inference or poisoning attacks. The intricate architecture of FL, involving numerous clients and frequent communication, inevitably expands the attack surface for adversarial threats.

In this academic work, we provide a comprehensive comparative analysis of several aggregation methods, including the novel evolutionary algorithm-based FedGA, to rigorously assess their robustness against a spectrum of federated learning threats. We systematically investigate and categorize prominent adversarial techniques such as label flipping, model scaling, and model inversion via GAN. Crucially, we reproduce and simulate these attacks within custom scenarios on a sophisticated hierarchical edge–fog–cloud topology. Our evaluation spans multiple datasets to thoroughly analyze the performance and resilience of the FedGA aggregator in adversarial environments. Our key contributions include a systematic exploration of existing vulnerabilities and the development of a reusable attack simulation framework. This framework is designed to serve as a vital benchmark for testing future aggregation strategies under realistic attack conditions.

This academic work is structured as follows:

- **Chapter I** introduces the fundamentals of machine learning, deep learning, and federated learning.
- **Chapter II** presents a comprehensive review of security and privacy threats in FL, classifying different types of attacks and vulnerabilities.
- **Chapter III** explores aggregation mechanisms and their role in mitigating adversarial behavior.
- **Chapter IV** details the experimental setup, implementation of selected attacks, and analysis of aggregator performance, particularly FedGA.
- Finally, a **general conclusion** summarizes the key findings and outlines future research directions.

Chapter 1: Fundamentals of Federated Learning

I.1. Introduction:

Data is key to industries, businesses, governments, and activities as a whole; without data, decision-making becomes challenging if not impossible. Data is very important for any business as it helps streamline management and gives one an understanding of operations. With the advent of Industry 4.0, the amount of data is huge and complex. To deal with this, machine learning (ML) has been a useful method as it has facilitated decision-making and learning more about the customer and operational trends. As a result, machine learning is a competitive technology that most organizations possess. In addition, the interaction between artificial intelligence (AI) techniques and intelligent cyber-physical systems (ICPS) has also underscored the importance of the technologies. In this chapter, we will explore artificial intelligence (AI) and discuss its relationship with machine learning (ML) and deep learning (DL). We will also look at different learning techniques, paying attention to machine learning. After which, we will delve into deep learning and explore artificial neural networks. Finally, we will introduce federated learning.

I.2. Artificial Intelligence (AI):

Artificial intelligence refers to systems that exhibit intelligence similar to that of humans. The programs are said to be emulated by the machines, allowing them to perform human tasks on behalf of the humans. These machines then must be computer-augmented to think, decide, perceive, and solve problems as are human beings.

In the macro picture of AI, Machine Learning, or ML, is an area within computer science that seeks to develop algorithms and statistical models that allow computers to learn from large amounts of data, without explicit programming. This machine learning finds its roots important as it provides various techniques that are further used in making different AI systems.

Deep learning ever since evolved as a particular step into ML. This subfield employs artificial neural networks with several layers to model extremely complex patterns in large datasets. It has achieved great results in image recognition, speech recognition, and natural language processing, which themselves have multiple levels of hierarchical features [1].

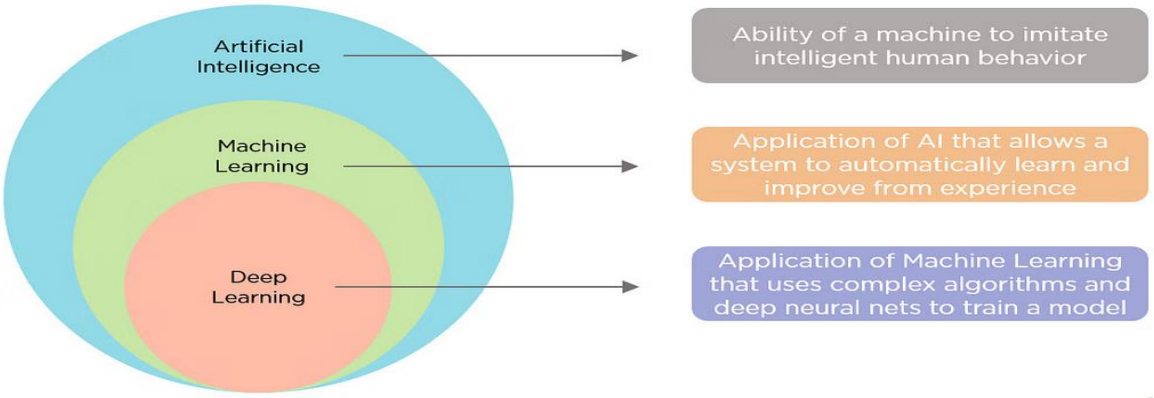


Figure I-1:relation between AI, ML and DL

I.3. Machine Learning:

Machine learning is part of artificial intelligence that aims to create methods and models that allow computers to learn and perform better, without being programmed for each task.

According to the classical definition by Tom Mitchell (1997), 'a computer program is said to learn from experience E with respect to a task T and a performance measure P, if its performance on T, as measured by P, improves with experience E.'[2]

I.3.1. Fundamental Concepts of Machine Learning:

- **Training Data:** A series of information principally employed to enlighten a machine learning framework. The collection usually entails samples and, in the event of supervised learning, the corresponding outputs.
- **Model:** A theory that explains the interplay of two or more variables, including input and output. Models have varied forms, ranging from linear equations to complex neural networks.
- **Parameters:** are internal variables that are modified during the training phase to decrease prediction loss.
- **Cost Function (or Loss Function):** This is the measure of the difference between the model's predictions and the actual values. The goal of learning is to minimize this function.
- **Optimization Algorithm:** A way that model parameters are regulated to lessen the cost function. Gradient descent and its variants are the most popular algorithms.
- **Generalization:** A machine learning model's ability to perform well on data it has not seen during training. This is the ultimate objective of building a model.
- **Overfitting:** can be described as the condition whereby a model has been able to learn from the training data to a point where it has learned it too well.
- **Underfitting:** A state of affairs, where a model's complexity cannot match the complexity of the data, such that it gives poor performance on the test data.

I.3.2. **Categories of Machine Learning:**

Machine learning can be categorized according to the nature of data available and the learning objective as follows:

Table I-1:Classification of Supervised, Unsupervised, and Advanced Learning Techniques

Type	Description	Applications
Supervised Learning	Learns from labeled data (input-output pairs). [3]	Classification, Regression
Unsupervised Learning	Discovers hidden patterns in unlabeled data. [3]	Clustering, Dimensionality Reduction
Reinforcement Learning	Learns via rewards/penalties from interactions.[3]	Robotics, Game AI
Semi-Supervised Learning	Uses both labeled and unlabeled data.	Speech Recognition
Transfer Learning	Reuses pre-trained models for new tasks.	Image Recognition, NLP
Federated Learning	Decentralized training across devices.	Privacy-preserving AI

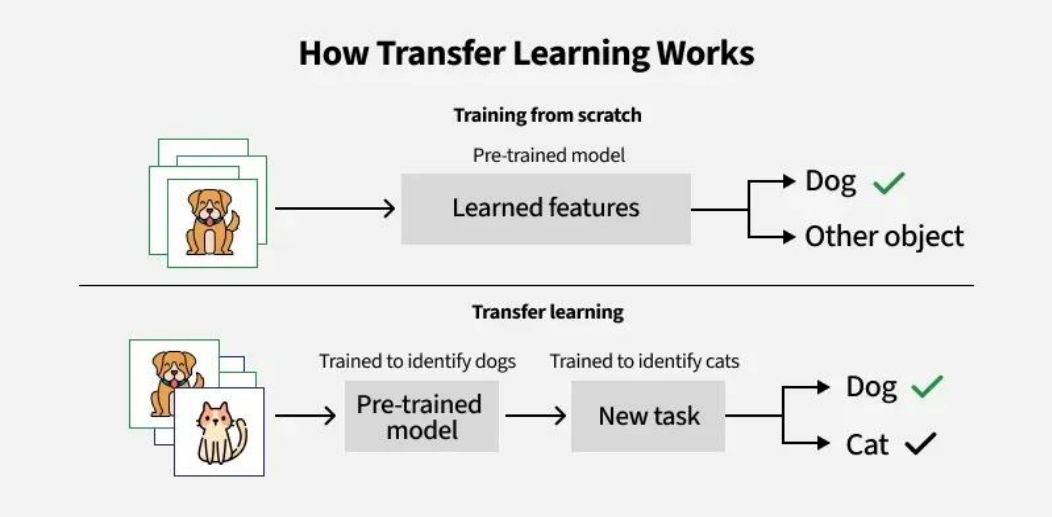


Figure I-2:the key stages of the Transfer Learning process

- **Federated Learning:** is a concept that allows a machine learning model to be trained across multiple devices or servers.

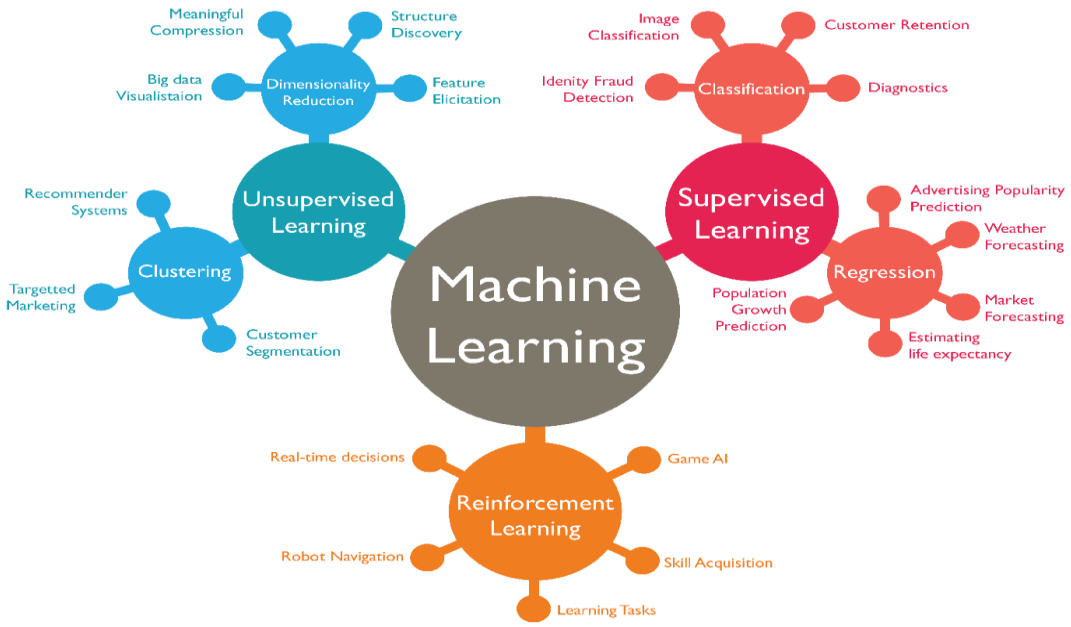


Figure I-3:Types of Machine Learning

I.4. Deep learning:

Deep learning is a kind of machine learning that concentrates on the design and utilization of artificial neural networks with numerous layers to imitate and solve complex problems. It employs hierarchical preview extraction, suggesting that lower layers can capture basic patterns, and higher layers can merge them to develop a better abstract description. These advancements in the field come in the form of vision, language understanding, and speech recognition [4].

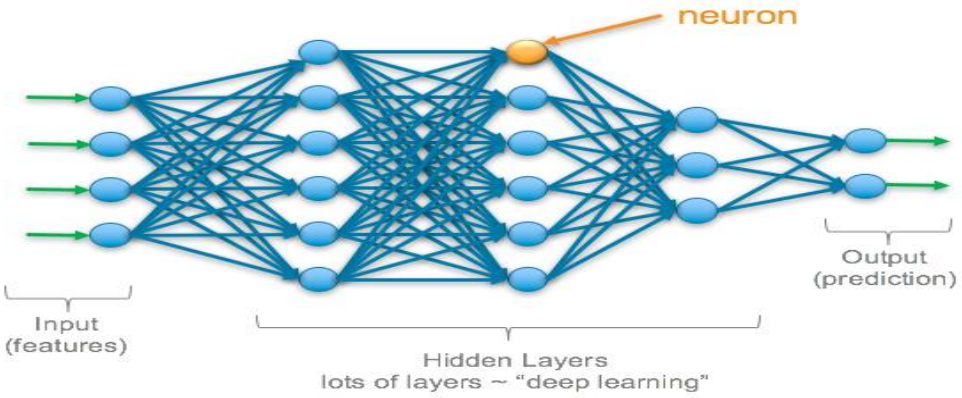


Figure I-4:Simple representation of A neural Network

I.4.1. **Single-Layer Perceptron:**

A single-layer perceptron is a type of linear classifier that is the simplest type of an artificial neural network. It consists of a single layer of output nodes that compute a weighted sum of the input features and threshold this sum to produce the output (usually the polarity). Though

limited to problems that are linearly separable, it can serve as good motivation for the analysis of more complicated neural network architectures [5], [6].

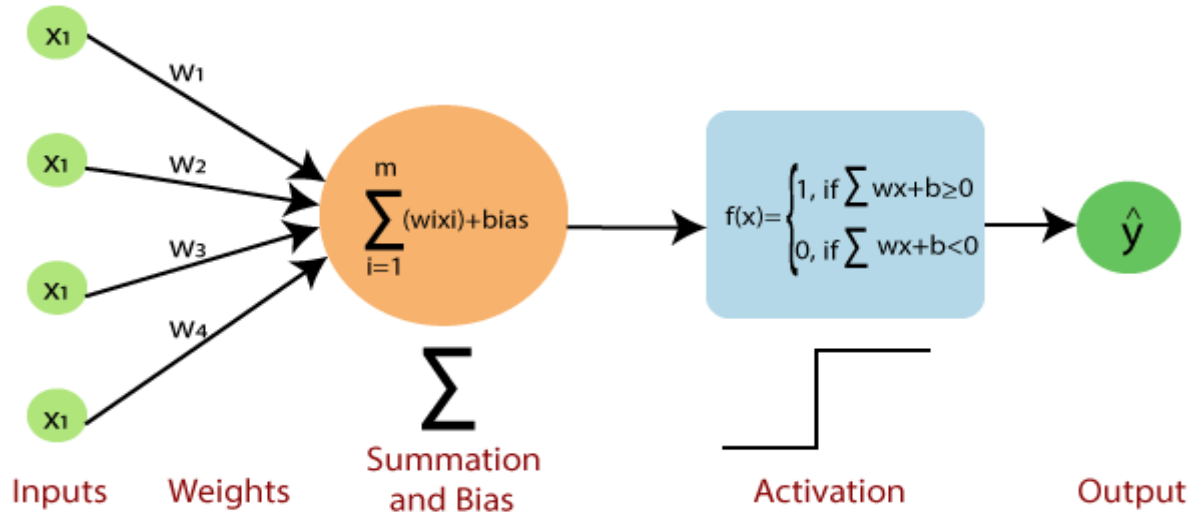


Figure I-5:Single Layer Perceptron representation.

I.4.2. **Multilayer Perceptron (MLP):**

A multilayer perceptron is a type of feed forward artificial neural network which consists of at least three layers of nodes. Besides the input layer and output layer, it also consists of one or more hidden layers. Each node is a neuron that uses a nonlinear activation function to the weighted sum of its inputs. Among the many tasks that it can complete are classifying and recognizing patterns [7].

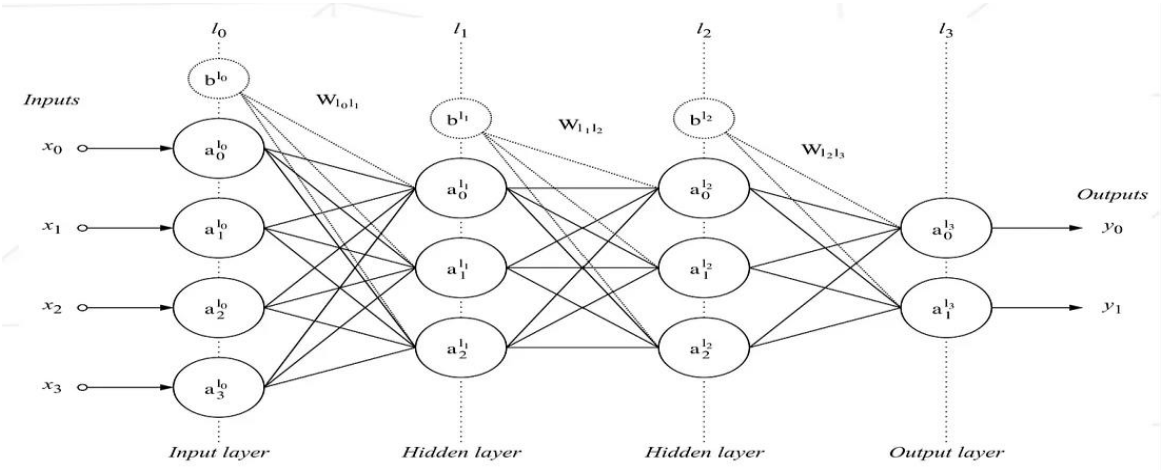


Figure I-6:Overview of a Multi Layers Perceptron.

I.4.3. **Activation fonctions:**

Activation functions are operations which are applied on the output of the neuron, in order to introduce non-linearity into the output of a neuron. These functions calculate whether a neuron should be activated or not by computing the weighted sum of the input signal and such input signals are then passed through each layer of the neural network. This provision introduces non-linear properties into the input-output mapping capabilities of a network.

Sigmoid Function:

The sigmoid function is defined as (I.1). This layer takes the input and transforms it into a range of 0 to 1, making it ideal for binary classification problems. However, it has trouble with vanishing gradients during backpropagation, which causes issues in deeper layers [3].

$$Sig(x) = \frac{1}{1-e^{-x}} \tag{I.1}$$

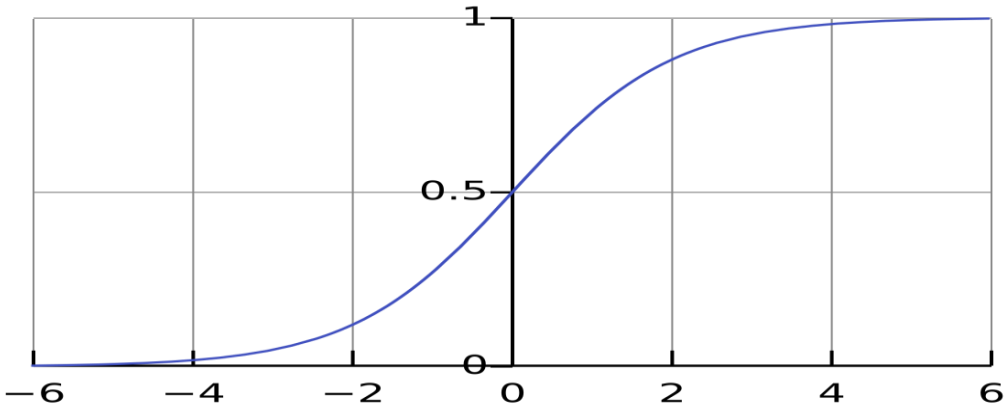


Figure I-7:Graphical representation of the sigmoid activation function.

Hyperbolic Tangent (Tanh):

The hyperbolic tangent function is defined as (I.2) Mapping the input to a range of -1, 1, and centering the output around 0 makes the tanh function slightly better during training than the sigmoid function, however, the tanh function has some issues with gradients.

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{I.2}$$

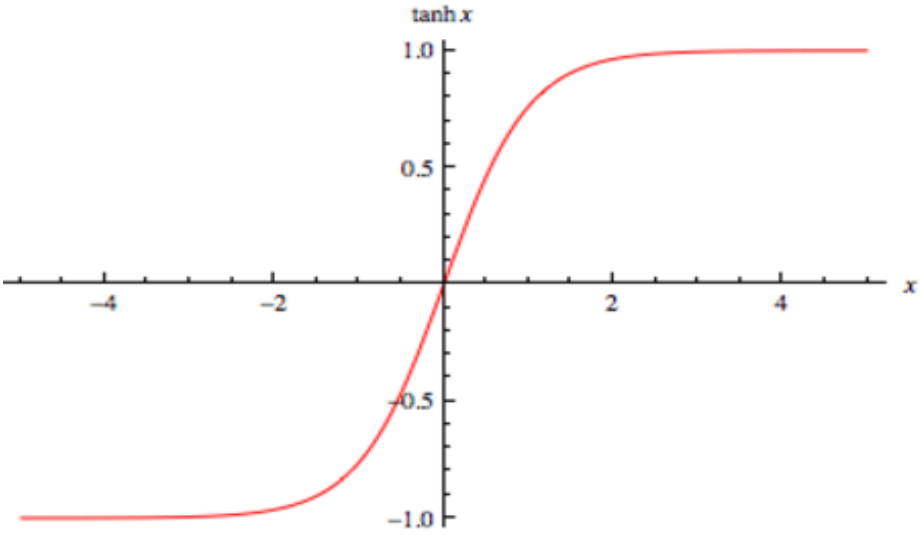


Figure I-8:Graphical representation of the hyperbolic tangent (tanh)

Rectified Linear Unit (ReLU):

The ReLU function is defined as (I.3). It gives zero for all negative inputs and returns the input for all positive inputs. ReLU has become one of the most popular activation functions in deep learning because of its simplicity and computational efficiency. It resolves the vanishing gradient issue but may cause neurons to stop functioning [3].

$$f(x) = max(0, x) \tag{I.3}$$

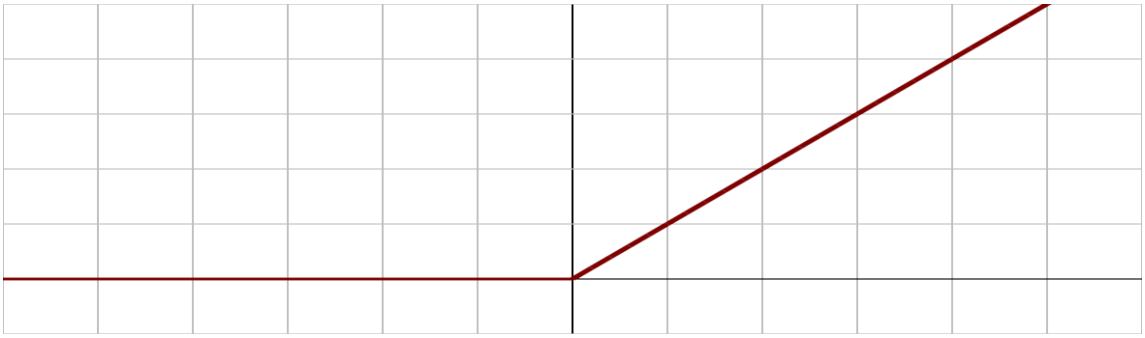
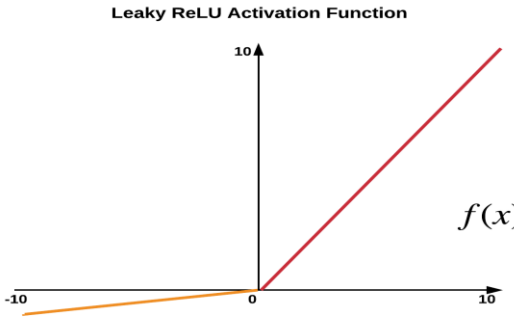


Figure I-9:Graphical representation of the rectified linear unit (ReLU)

Leaky ReLU:

A version of ReLU known as Leaky ReLU solves the issue of dying ReLU neurons in neural networks. The function adds a slight negative slope for inputs that are less than zero and it appears as (I.4). where α a small constant (e.g., 0.01). Even negative inputs will provide a small contribution to the learning process because of this design.

$$f(x) = \begin{cases} x, & if\ x \geq 0 \\ \alpha x, & if\ x < 0 \end{cases} \tag{I.4}$$



$$f(x) = \begin{cases} 0.01x & \text{for x} < 0 \\ x & \text{for x} \geq 0 \end{cases}$$

Figure I-10:Graphical representation of the leaky rectified linear unit (Leaky ReLU)

Softmax Function:

The output layer of a neural network for multi-class classification tasks usually implements the softmax function. The softmax function transforms raw scores (logits) into probabilities by making sure the cumulative sum of all outputs reaches 1. The mathematical representation of the softmax function is described as (I.5). The softmax function computes the probability distribution among multiple classes by converting raw scores into probability values. The normalized [3].

$$Softmax(z_j) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad \text{(I.5)}$$

I.4.4. **Deep Learning Optimizers:**

Deep learning optimizers function as algorithms which modify neural network parameters through training to reduce the loss function. The algorithms work by utilizing gradients computed through backpropagation to continuously adjust model parameters which leads to efficient convergence toward optimal solutions. The selection of optimizer has a substantial effect on both the rate of learning and the stability and generalization ability of deep learning models. The design of deep learning models depends heavily on the optimizer chosen for their training process. The optimizer serves as a critical component of deep learning development because it determines the learning rate and approach to update model parameters.

Batch Gradient Descent (BGD):

Batch Gradient Descent (BGD) computes the gradient of the loss function with respect to the parameters using the entire training dataset. This ensures precise updates but can be computationally expensive for large datasets. The update rule is given by (I.6) where $W^{(t)}$ is the average loss over all training samples? While BGD guarantees convergence to the global minimum for convex loss functions, its high computational cost limits scalability.

$$W^{(t)} = W^{(t-1)} - \alpha \frac{\partial J(W^{(t-1)}, T)}{\partial W} \quad \text{(I.6)}$$

Stochastic Gradient Descent (SGD):

Stochastic Gradient Descent (SGD) updates the model parameters using the gradient of the loss function with respect to a single data point or a mini-batch. Mathematically, it is expressed as (I.7) where $\boldsymbol{\eta}$ is the learning rate, $\boldsymbol{\theta}$ represents the parameters, and J is the loss function. While simple and effective, SGD can suffer from slow convergence and sensitivity to hyperparameter tuning [8].

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\eta} \cdot \boldsymbol{\nabla_{\theta} J}\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \quad \text{(I. 7)}$$

Mini-batch Gradient Descent (MBGD):

The Mini-batch Gradient Descent (MBGD) method combines BGD with SGD through its processing of gradient calculations on training data subsets. The update method functions just like SGD except it processes training data through batch operations instead of individual sample computations:

$$\theta = \theta - \boldsymbol{\eta} \cdot \nabla_0 J\left(\theta; X^{(b)}, Y^{(b)}\right) \quad \text{(I. 8)}$$

The combination of SGD efficiency with BGD stability makes MBGD the most frequently implemented optimization technique.

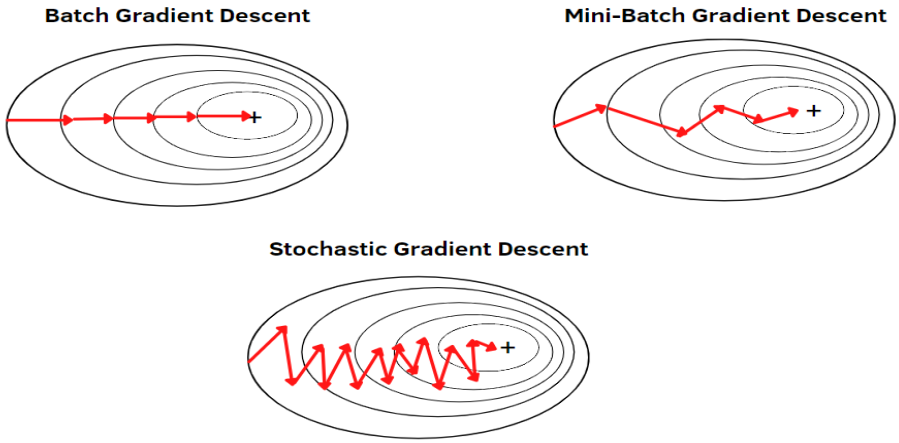


Figure I-11:Difference between batch GD,Stochastic GD,mini-batch GD

Momentum:

The Stochastic Gradient Descent (SGD) algorithm achieves improved performance through the application of momentum which utilizes a velocity term to aggregate historical gradients and decrease oscillations while accelerating convergence. The update equation uses (I.9) the momentum coefficient $\boldsymbol{\gamma}$ as shown in this rule. The method produces a better-optimized trajectory particularly for areas of the function with sharp curves and uncertain gradient variations [8].

$$v_t = \boldsymbol{\gamma v_{t-1}} + \eta \nabla_{\theta} \mathrm{J}(\theta) \text{ and } \theta = \theta - v_t \quad \text{(I.9)}$$

Root Mean Square Propagation (RMSProp):

AdaGrad has its learning rate decrease over time and RMSProp solves this problem through its squared gradient moving average. The update rule is (I.10) where ϵ serves as a small constant to prevent division by zero. RMSProp adapts well to non-stationary objectives and delivers consistent update stability [8], [9].

$$v_t = \boldsymbol{\gamma v_{t-1}} + (\mathbf{1} - \boldsymbol{\gamma})\left(\boldsymbol{\nabla_\theta J(\theta)}\right)^{\mathbf{2}} \text{and } \theta = \theta - \frac{\eta}{\sqrt{v_t + \epsilon}} \nabla_\theta J(\theta) \quad \text{(I.10)}$$

Adaptive Moment Estimation (Adam):

Adam presents the combined advantages of adaptive learning rates from AdaGrad and RMSProp together with momentum-based updates. The algorithm calculates customized learning rates for every parameter through the combination of first-order (mean) and second-order (uncentered variance) gradient moments. The update rule consists of an initial formula (I.11) which then includes bias correction before implementing parameter changes. The Adam optimizer obtained widespread recognition for its strong performance in multiple applications [8].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta) \text{ and } v_t = \beta_2 v_{t-1} + (1 - \beta_2)\left(\nabla_\theta J(\theta)\right)^2 \quad \text{(I.11)}$$

I.4.5. **Deep Learning Types:**

Below are some commonly used types of deep learning models, along with their characteristics and graphical representations:

Convolutional Neural Networks (CNN):

Specialized architectures called Convolutional Neural Networks (CNNs) work particularly well when processing grid-type information such as images. CNNs extract spatial feature hierarchies using convolutional layers with filters and perform dimensionality reduction with pooling layers. CNNs demonstrate exceptional performance in computer vision applications because they can detect patterns at local levels while maintaining translation independence.

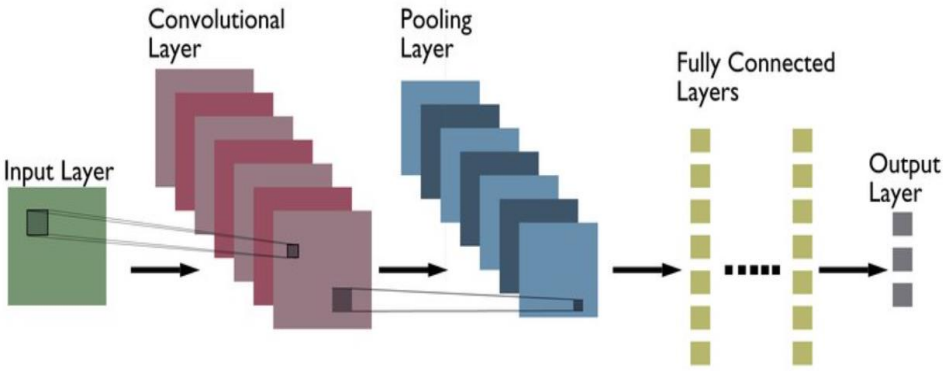


Figure I-12:CNN Architecture

Recurrent Neural Networks (RNN):

Sequential data processing in Recurrent Neural Networks (RNNs) works through a hidden state that memorizes information from past inputs. RNNs prove effective for learning temporal relationships which enables them to perform tasks like time series prediction and language modeling and machine translation. The conventional RNN architecture faces issues with gradient vanishing and exploding during training operations [10].

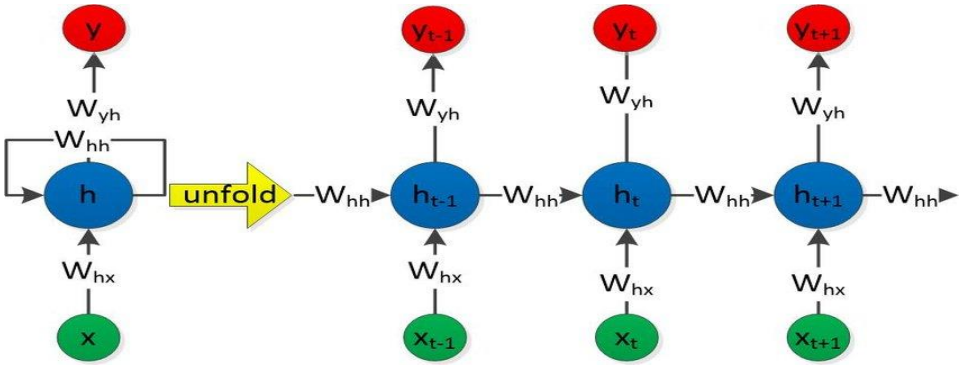


Figure I-13:RNN Architecture

Long Short-Term Memory Networks (LSTM):

RNNs developed the Long Short-Term Memory Networks (LSTMs) to solve the vanishing gradient problem. The LSTM architecture includes memory cells and gating mechanisms that consist of input and forget and output gates which control information flow for detecting extended relationships in time-series data. The LSTM architecture finds extensive application in speech recognition systems alongside sentiment analysis tasks [10].

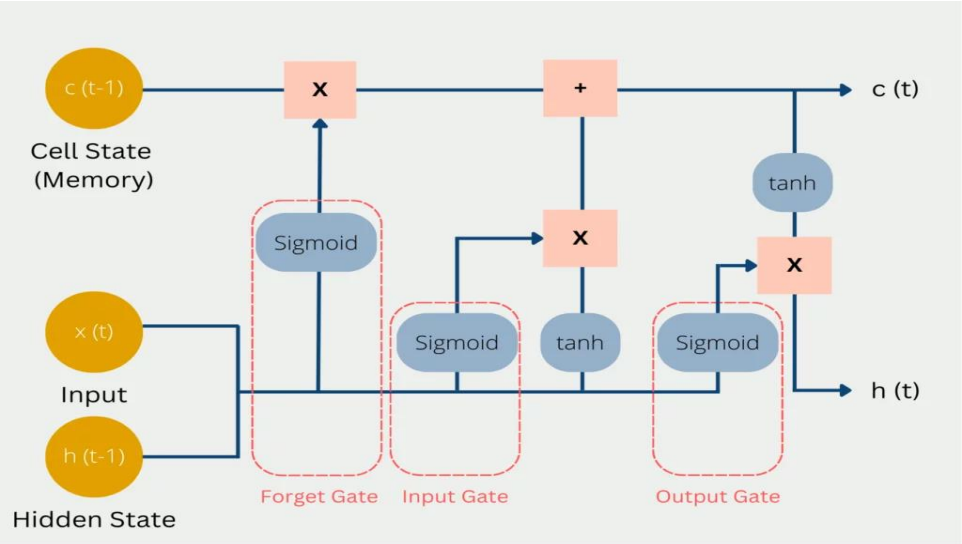


Figure I-14:diagram illustrates the architecture of an LSTM model

Autoencoders:

An unsupervised neural network which performs dimensionality reduction and feature extraction is known as an autoencoder. An autoencoder includes two main components which are the encoder that compresses data into a latent space and the decoder which recreates the original input. The applications of autoencoders extend to both anomaly detection and denoising as well as generative modeling [3].

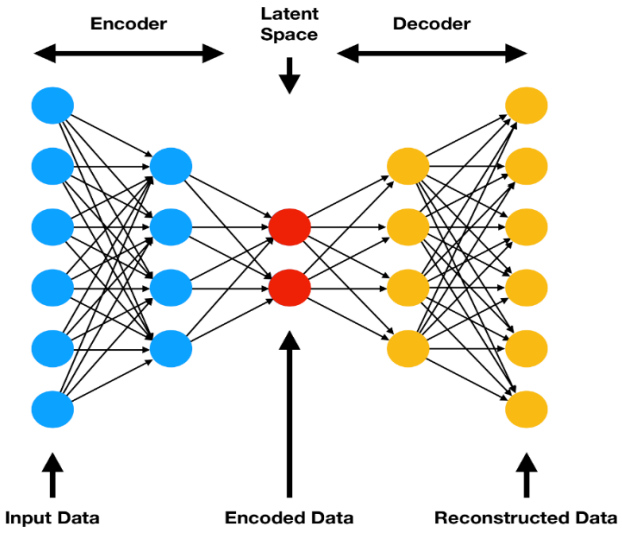


Figure I-15:Autoencoder Architecture

I.5. Federated learning:

I.5.1. Definition:

Decentralized machine learning under the Federated Learning (FL) framework allows distributed clients to train models together without sending their raw data to each other. Federated Learning distributes the training process across clients who work independently on their local data before sending model updates such as gradients or parameters to the central orchestrator or across peer-to-peer networks. Distributed systems thrive under this approach because it places privacy preservation and computational efficiency and scalability at the forefront. The following principles establish the core of this model:

Data Decentralization:

The system keeps data local by storing original datasets on user equipment. The system minimizes privacy dangers from data concentration while lowering communication expenses to meet both regulatory requirements and edge computing resource boundaries [11].

Collaborative Model Refinement:

When a global model receives feedback from various users, these clients work together to enhance its performance through their decentralized approach. The global model uses collective participant knowledge to improve its capabilities yet each client maintains complete control over their local data which builds trust and promotes multi-stakeholder compliance [12].

Global and Local Model Dynamics:
The Federated Learning (FL) process relies on two intertwined basic elements:

- **Global Model**: Through iterative local model updates combined by federated averaging, the global model represents collective knowledge obtained from distributed data sources.
- **Local Models**: These models receive training from individual clients using their specific datasets that show diverse variations in dimensions and distribution patterns and feature structures.

Heterogeneous Data Challenges: FL systems encounter statistical heterogeneity through non-IID client data distributions. The non-IID nature results from different ways data gets collected and from variations in user conduct and environmental states which require strong aggregation methods along with dynamic optimization approaches to achieve convergence and model equity [13].

I.5.2. Federated Learning Training Process:

The training process for federated models includes multiple sequential operations which together create a training round in federated learning.

1. **Initialization:** The central server creates the global model by either utilizing random parameters or using an existing pre-trained model during initialization. This initial process occurs solely at the start of the training process.
2. **Client Selection:** The server chooses a group of clients during each training round to be part of the training process. The server uses random selection as well as criteria related to client availability and their past reliability and data quality to determine which clients will participate. Occasionally, the system may choose all existing clients to take part in the process [14], [15].
3. **Model Distribution:** The server delivers the current global model to the chosen clients through distribution. The sending process involves the whole model delivery or only transmitting parameter modifications since the last round according to the system's architecture.
4. **Local Training:** Every selected client executes training operations with the model it received based on its local data until either a specific number of epochs or a predefined convergence criterion is satisfied. The standard training algorithms include Stochastic Gradient Descent (SGD) and its related versions.
5. **Update Calculation:** Clients perform the update calculation after completing their local training to determine the parameter differences between trained and original server parameters.
6. **Transmission of Updates:** Clients transmit their update information to the central server for processing. Updates undergo preprocessing through compression techniques which decrease the necessary bandwidth or through differential privacy methods for privacy protection.
7. **Aggregation:** The server performs update aggregation by processing the received updates to generate a global update. The aggregation method depends on FedAvg (Federated Averaging) which utilizes weighted averages of updates based on local dataset sizes [16].
8. **Global Model Update:** The server processes the combined model modifications through a standard gradient descent-based update rule to update the global model framework.
9. **Evaluation:** The server has the option to test the updated model by running it on a validation dataset which could be present. The model training process runs multiple iterations until either the model reaches convergence or the predetermined number of iterations is achieved. The trained global model becomes operational after the completion of its training process. The approach for performing this process varies depending on the specific system architecture in use. Under specific implementations, clients can upload model parameters directly to servers instead of update data, and servers have alternative methods to modify models through pruning and model distillation before releasing them to clients.

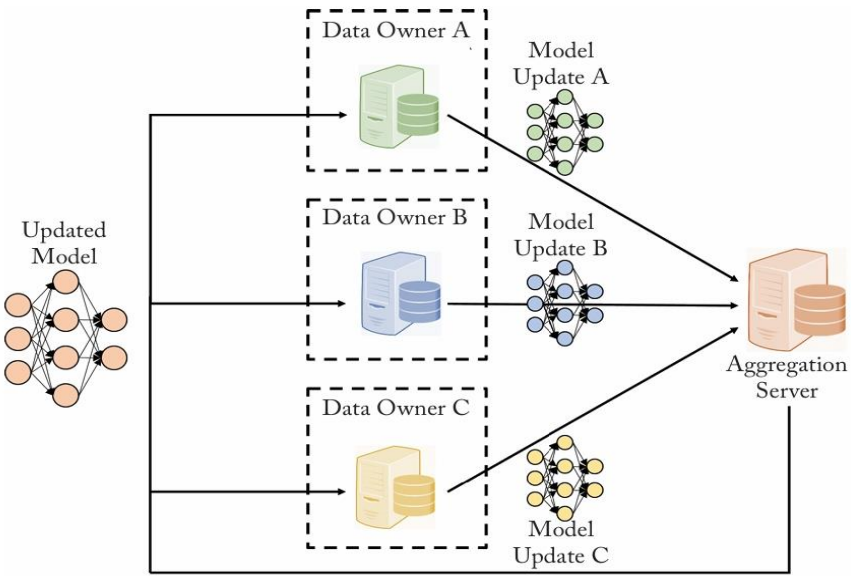


Figure I-16:An example federated learning architecture: client-server model.

I.5.3. **Categories of Federated Learning:**

Federated Learning (FL) can be categorized into three main paradigms based on the nature of data distribution across participating entities: Horizontal Federated Learning, Vertical Federated Learning, and Federated Transfer Learning. These classifications reflect how data samples and features are shared or not shared among participants, and they determine the architectural design and methodological approaches used in FL systems.

I.5.3.1. ***Horizontal FL:***

Horizontal FL refers to scenarios in which participating clients share models with **similar feature spaces but different data samples**. In other words, the same set of attributes is observed across different users or devices, but the individual instances vary. This type of FL is commonly applied in settings such as mobile computing, where multiple users interact with similar applications on their smartphones but generate distinct usage data [17].

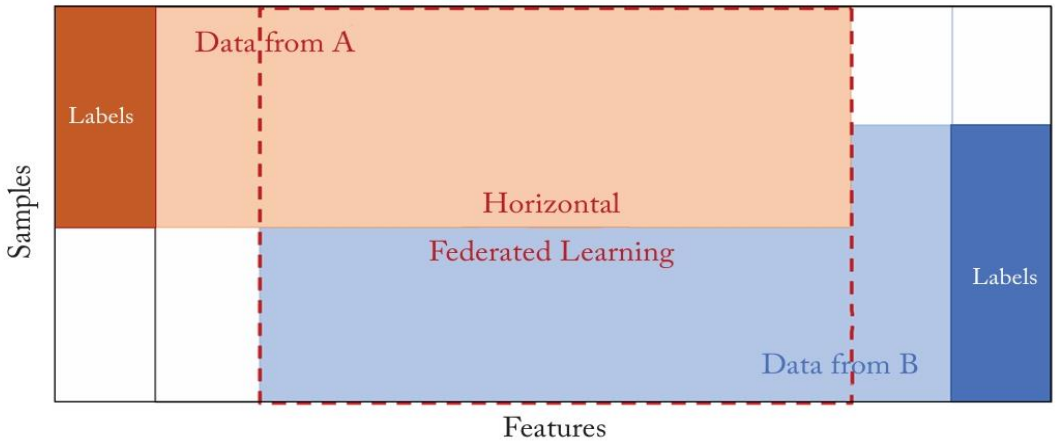


Figure I-17:illustration of Horizontal federated learning[17]

I.5.3.2. *Vertical FL:*

In contrast, **Vertical FL** applies when clients share **the same set of data samples** but possess **different feature sets**. This scenario typically occurs in cross-organizational collaborations, such as between a bank and an e-commerce platform, where both may hold different types of information about the same set of customers (e.g., transaction history vs. browsing behavior) [17], [18].

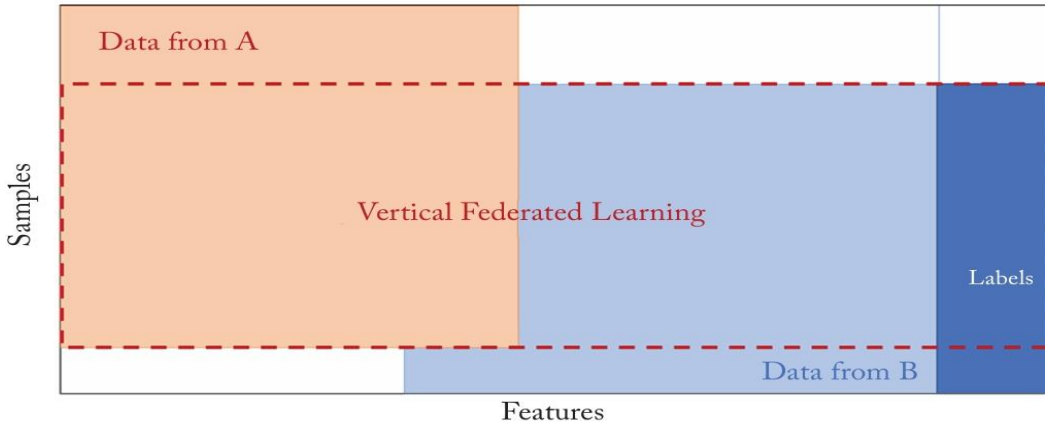


Figure I-18:illustration of Vertical federated learning[18]

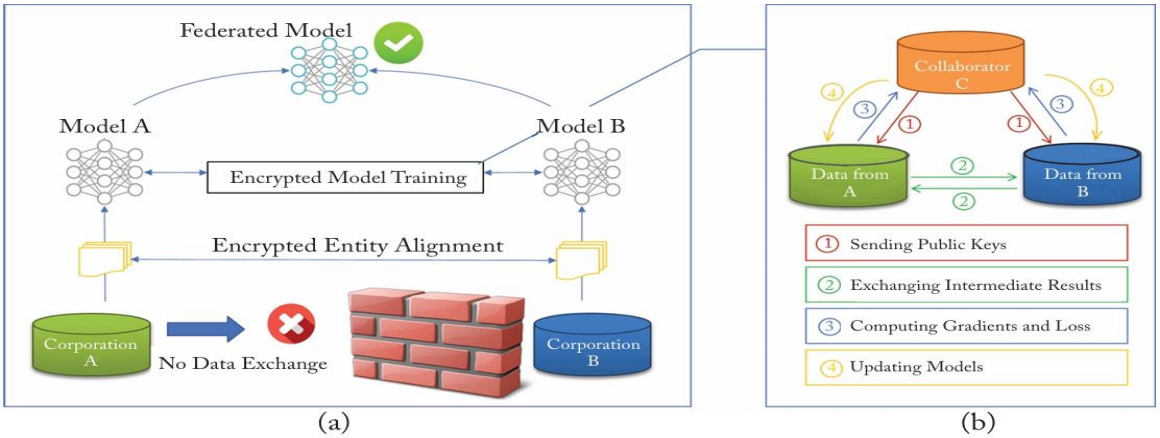


Figure I-19:Architecture for a vertical federated learning system

I.5.3.3. *Federated transfer learning:*

Finally, **Federated Transfer Learning (FTL)** is employed in situations where there is minimal overlap in both samples and features among participants. FTL leverages transfer learning techniques to enable knowledge sharing across heterogeneous data distributions, making it particularly useful in highly decentralized environments with limited data commonality [19].

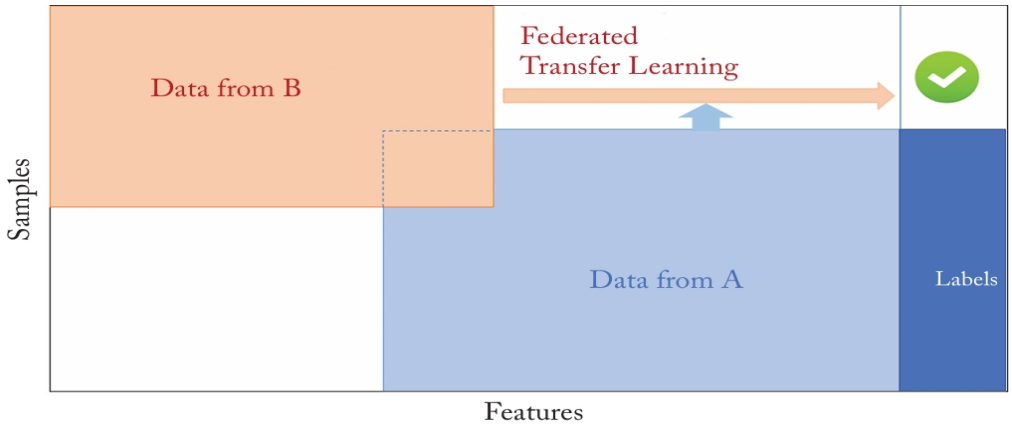


Figure I-20:illustration of Federated transfer learning[18]

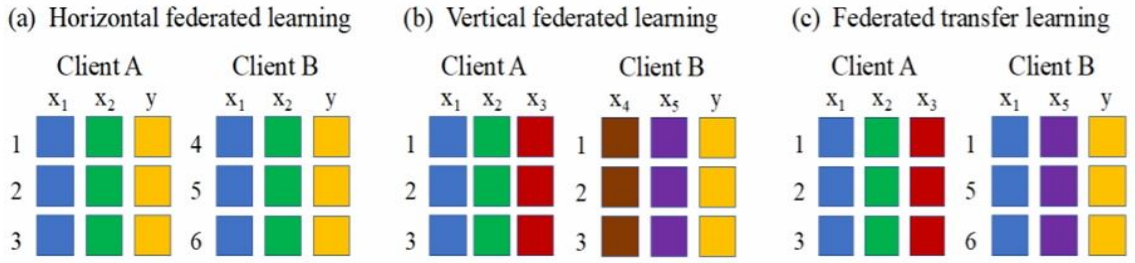


Figure I-21:Difference between the three Categories

I.5.4. **Degree of federated Learning:**

Federated Learning (FL) can also be classified based on the **scale and nature of participating devices or organizations**, leading to two primary deployment paradigms: **Cross-Silo Federated Learning** and **Cross-Device Federated Learning**. These categories reflect the differences in infrastructure, data distribution, and communication constraints within FL systems.

Cross-Silo Federated Learning:

Cross-Silo FL refers to scenarios involving a **limited number of well-defined, powerful participants**, such as hospitals, financial institutions, or enterprises, that collaborate to train a shared global model using their local data. These entities typically have access to relatively large datasets and sufficient computational resources. This setup is commonly used in organizational collaborations where privacy and regulatory compliance are key concerns [20].

Cross device Federated Learning:

In contrast, Cross-Device FL operates at a much **larger scale**, involving potentially millions of decentralized edge devices—such as smartphones, wearables, or IoT devices—each contributing to the learning process. In this paradigm, individual devices hold only a small amount of data and possess limited computational capabilities compared to cross-silo

participants. The system must therefore account for challenges such as unreliable network connectivity, device heterogeneity, and resource constraints [20].

I.5.5. **Network topology:**

I.5.5.1. ***Centralized FL:***

The centralized federated learning (FL) framework illustrated in Figure I-16 is characterized by a hierarchical architecture in which a central server orchestrates the training process. During the initial phase (Step 1), the server collects locally trained models from participating client devices. In the subsequent phase (Step 2), the server performs model aggregation to synthesize a global model, which is then disseminated back to all clients. This paradigm employs a star-shaped network topology, comprising a single coordinating server and a distributed network of client nodes, thereby enabling centralized coordination of decentralized learning processes [19].

I.5.5.2. ***Peer to Peer Decentralized Federated Learning Architecture:***

The fully decentralized federated learning (FL) framework depicted in Figure I -22 eliminates the need for a central server to coordinate model aggregation, instead adopting a peer-to-peer (P2P) communication architecture for inter-node collaboration. In this paradigm, participating nodes directly exchange model updates and perform consensus-based aggregation without reliance on a centralized authority. By decentralizing the communication and aggregation processes, this approach mitigates potential risks associated with centralized FL, such as single points of failure, susceptibility to server malfunctions, and security vulnerabilities inherent to centralized coordination. The resulting system enhances robustness and resilience by distributing computational and communicative responsibilities across all nodes in the network [19].

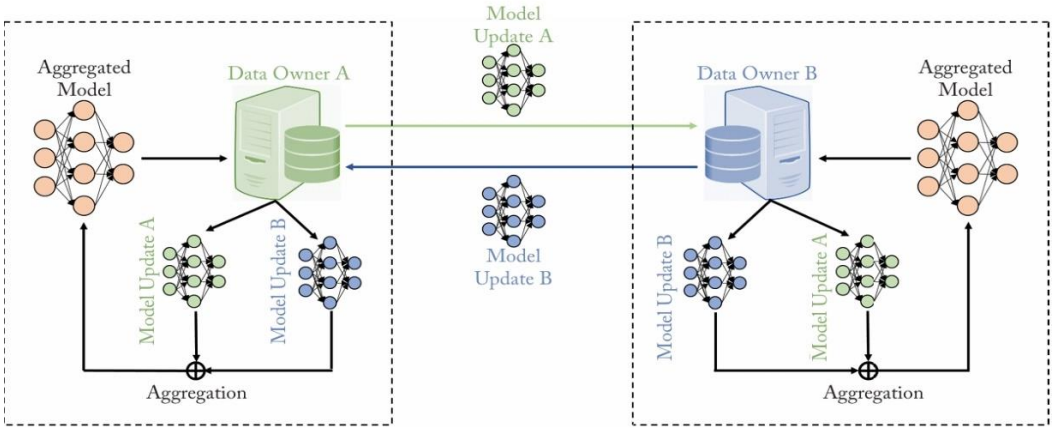


Figure I-22:An example federated learning architecture: peer-to-peer model [18].

I.5.5.3. ***Hierarchical topology:***

In a hierarchical topology, nodes are organized into layers or hierarchical clusters, where each level performs local data aggregation before propagating updates to higher layers. Lower-

level devices (such as IoT sensors or smartphones) send their data to intermediate nodes known as edge servers, which carry out preliminary processing and aggregation. The consolidated information is then forwarded to a central cloud layer responsible for global model updates or strategic decision-making. This multi-tiered architecture enables efficient data consolidation and distributed processing, as seen in smart city infrastructures where edge servers aggregate local traffic sensor data before transmitting summarized updates to a central cloud platform for city-wide analysis and optimization [22].

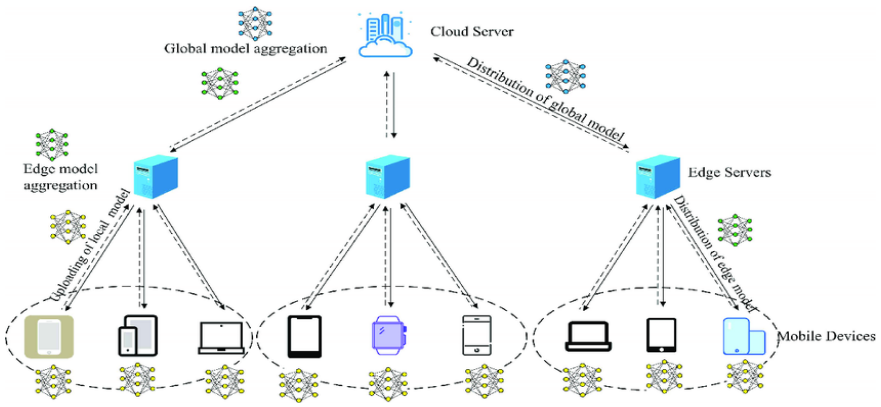


Figure I-23: *Hierarchical Federated Learning Framework*

I.5.5.4. ***Hybrid Topology:***

A hybrid topology integrates components of multiple architectural paradigms, synthesizing centralized and decentralized frameworks to optimize system performance. For instance, such a topology may primarily adopt a centralized communication model for core operations while incorporating decentralized mechanisms to address specialized tasks or adapt to dynamic contextual requirements. This dual approach balances the efficiency of centralized control with the resilience and flexibility inherent in decentralized systems. An illustrative example includes leveraging centralized coordination for routine data processing while delegating fault-tolerant consensus protocols to decentralized subnetworks during node failures or high-latency scenarios. By strategically combining these elements, hybrid topologies enhance adaptability, scalability, and robustness in heterogeneous computing environments [20].

Figure I-24:Types of Federated Learning based on System Architecture

I.5.6. **Applications of Federated Learning:**

Federated Learning (FL) has emerged as a transformative paradigm across diverse domains, enabling collaborative model training while preserving data privacy and addressing logistical challenges associated with centralized data aggregation. Below is an analysis of its applications in key sectors:

Healthcare:

FL enables cross-institutional collaboration in health, enabling in-hospital and research agencies to collaboratively train predictive models (e.g., for disease diagnosis or prognosis), without needing to share the underlying sensitive patient records. For instance, federated systems allow cancer detection models to be built without sharing data, as hospitals extract knowledge from their local datasets, preserving GDPR and HIPAA compliance. This methodology not only addresses the potential risk of data breaches but also facilitate advancement of personalized medicine using heterogeneous data integration [21].

IoT and Edge Computing:

In IoT ecosystems, FL supports on-device learning for distributed smart devices, such as wearables and industrial sensors. By training models locally on edge nodes, FL reduces latency and bandwidth consumption while enhancing real-time decision-making. For example, predictive maintenance models in manufacturing environments leverage FL to aggregate knowledge from geographically dispersed sensors, improving fault detection accuracy without centralizing raw operational data [22].

Finance:

Financial institutions employ FL to address fraud detection and risk management challenges while adhering to stringent data governance policies. Banks and payment processors collaboratively train transaction monitoring models using decentralized datasets from local servers, ensuring compliance with jurisdictional data residency requirements. This decentralized approach enables the detection of anomalous patterns across institutions without exposing proprietary or customer-specific transactional data [22].

Mobile Computing:

FL underpins personalized user experiences in mobile applications, such as next-word prediction in smartphone keyboards (e.g., Google's Gboard). By training language models on decentralized user interactions, FL ensures that sensitive typing data remains on individual devices. This paradigm balances personalization with privacy, enabling scalable model updates across millions of devices without centralized data collection.

Autonomous Systems:

In autonomous vehicle networks, FL enables vehicles to collaboratively refine driving models using on-board sensor data (e.g., LiDAR, camera feeds) while preserving user privacy. For instance, decentralized learning frameworks allow vehicles to share model updates derived

from local driving conditions, enhancing collective situational awareness without transmitting raw sensor data to centralized servers. This approach accelerates the development of robust perception and navigation systems.

I.5.7. Advantages of Federated Learning Over Traditional Machine Learning:

Federated learning (FL) presents a paradigm shift in machine learning, offering distinct advantages over conventional centralized approaches, particularly in privacy, efficiency, scalability, and regulatory compliance [23].

Enhanced Privacy Preservation:

FL ensures data locality by retaining raw data on client devices, eliminating the need for direct data transmission to a central server. This decentralized architecture inherently mitigates privacy risks associated with centralized data aggregation, as sensitive information never leaves the source device. Techniques such as differential privacy and secure aggregation further amplify protection, aligning with modern privacy expectations.

Reduced Communication Overhead:

Unlike traditional methods that require transferring voluminous raw datasets, FL transmits only compact model updates (e.g., gradient parameters). This significantly reduces bandwidth consumption and latency, enabling efficient training across distributed networks. The communication cost scales sublinearly with the number of clients, making it particularly advantageous for large-scale applications.

Scalability in Heterogeneous Environments:

FL's decentralized framework supports integration with thousands of heterogeneous clients, accommodating diverse hardware capabilities and data distributions. By leveraging edge computing resources, it enables robust model training in resource-constrained settings (e.g., IoT devices, mobile networks) while maintaining performance consistency across varied computational environments.

Regulatory Compliance:

FL inherently aligns with stringent data protection regulations (e.g., GDPR, HIPAA) by design. Its privacy-preserving data handling practices—such as minimizing data exposure and enabling on-device processing—directly address legal requirements for data minimization and user consent. This compliance-by-design approach reduces organizational liability and fosters trust in data-driven ecosystems.

I.6. Conclusion:

This chapter introduced the fundamental concepts of artificial intelligence (AI), machine learning (ML), and deep learning (DL), highlighting their interdependence and their crucial role

in processing large and complex datasets in the era of Industry 4.0. Various ML approaches were discussed, including supervised, unsupervised, reinforcement, semi-supervised, transfer, and federated learning. Deep learning was also explored in detail, with its key architectures such as convolutional neural networks (CNN), recurrent neural networks (RNN), long short-term memory (LSTM) networks, and autoencoders, which are used to model complex patterns in data for applications like image recognition and natural language processing.

Finally, federated learning was presented as an innovative approach that enables decentralized model training while preserving data privacy. Its architecture, categories (horizontal, vertical, and transfer learning), network topologies (centralized, peer-to-peer, hierarchical, hybrid), and applications in sectors such as healthcare, finance, IoT, mobile computing, and autonomous systems were analyzed. The chapter concludes by emphasizing the advantages of federated learning, such as enhanced privacy, reduced communication costs, scalability, and compliance with data regulations. This sets the stage for the next chapter , which will explore different types of attacks related to privacy and security in machine learning and federated learning systems.

Chapter 2: Security Vulnerabilities in Federated Learning

II.1. Introduction:

This chapter explores the security threats that emerge in this framework after reviewing the fundamental learning principles studied in the opening chapter. The security model of federated learning demonstrates strong advantages regarding privacy protection and decentralized operations but creates distinctive weaknesses that endanger both data security and system stability. This chapter provides an examination of poisoning attacks and infiltration as well as communication failures to evaluate their potential consequences and the safety risks they present to federated learning systems.

II.2. Security Threats:

Despite its design to preserve data privacy by avoiding centralized data sharing, Federated Learning (FL) remains vulnerable to various security threats that can compromise the integrity**,** availability, and performance of the global model. These threats primarily stem from three components of the system: the central server, the clients, and the communication channels. A compromised central server may manipulate the aggregation process using malicious algorithms, undermining the global model's reliability. Malicious clients can perform model poisoning attacks by submitting falsified updates to distort learning outcomes or embed backdoors. Additionally, unsecured communication channels expose the system to eavesdropping**,** data interception, and message tampering**,** which may lead to information leakage or corrupted model updates [25], [26], [27], [28].

II.3. Poisoning Attacks:

Poisoning attacks are one of the most severe threats to federated learning systems. These attacks aim to weaken the global model's integrity by submitting some data or updates maliciously into the training procedure.

II.3.1. **Data poisoning:**

Data poisoning attacks represent a significant security challenge in federated learning, where malicious participants manipulate their local training data to compromise the global model. Unlike centralized machine learning where data can be inspected and vetted before training, federated learning's distributed nature makes data validation inherently difficult, creating opportunities for this attack vector [24].

II.3.1.1. ***Mechanisms of Data Poisoning:***

Data poisoning in federated learning may be accomplished using various techniques [25]:

Label Flipping: The adversary intentionally mislabels training examples; for example, labeling malicious emails as benign and vice versa. It can, of course, lead a model to learn wrong decision boundaries, thus yielding more general erroneous decisions or developing specific vulnerabilities [26], [27].

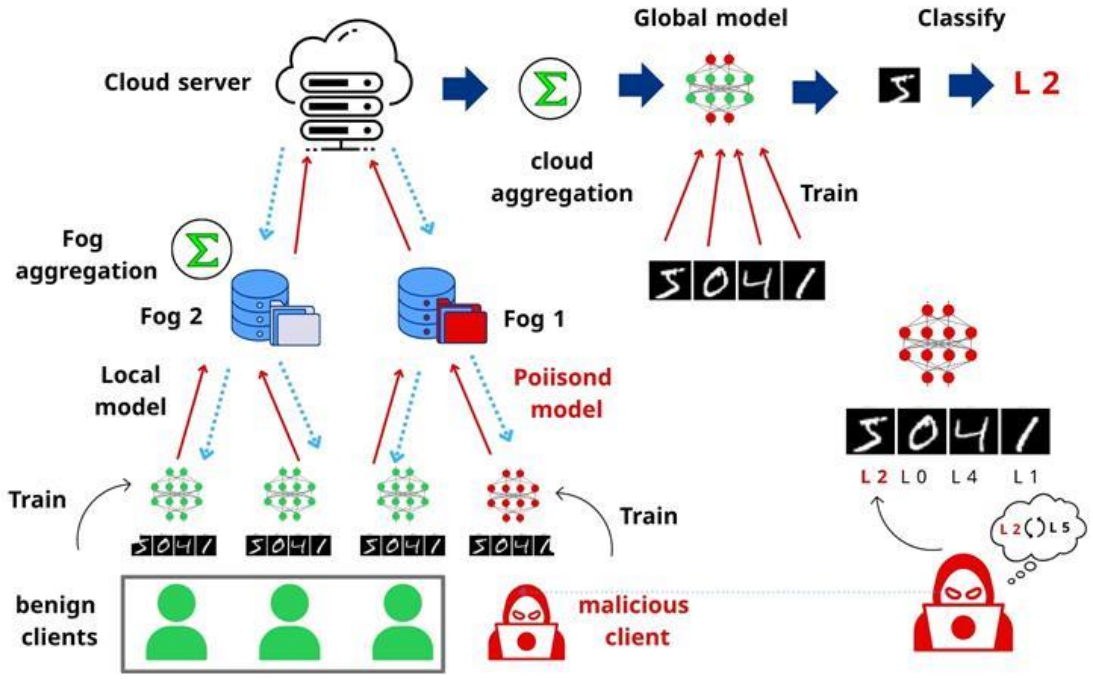


Figure II-1: *Label Flipping Attack Illustration*.

Feature Manipulation: The adversary influences the model's learning by perturbing feature values of instances in their training data instead of changing labels. This can be more subtle than label flipping and might be more challenging to detect.

Clean-Label Poisoning: An advanced version of poisoning in which the adversary does not alter labels but carefully creates input examples that look legitimate but cause the model to learn incorrectly. This is especially hard to detect, as the poisoned samples look "clean" under scrutiny.

Backdoor Insertion: The adversary attaches a trigger or pattern to certain entries of their training set and assigns these samples to a target class. This will, in turn, train the model to associate the trigger with the target class, thus creating a backdoor for onward exploitation [28].

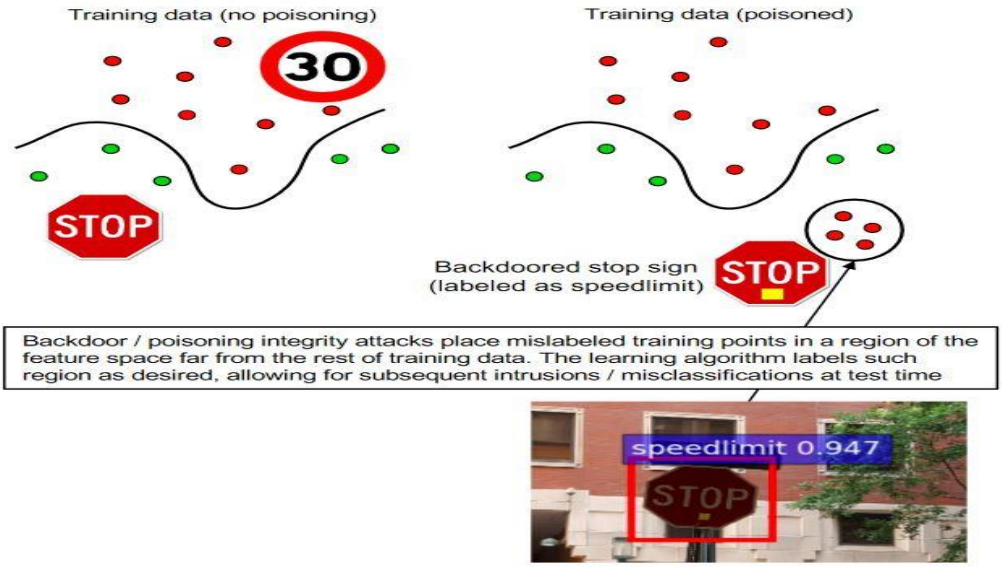


Figure II-2: Backdoor Attack in Data Poisoning

II.3.1.2. *Attack Objectives*

The types of damage-causing attacks depend on their objective, which are as follows [65]:

Untargeted (Random) Poisoning: Does not leave the overall effect of the model performances on every input, thereby making it useless for all its participants. This may be the form of attack that is less inconvenient to implement; nevertheless, it could easily be detected by performance monitoring.

Targeted Poisoning: Having good comportment in every class of inputs or at times may even throw at them but misclassifies definitely bringing some other specific ones within certain selected classes. As of now, the whole accuracy of the model remains unaffected, which grants this attack substantial time before it gets detected.

Backdoor Poisoning: That is making the model accept and behave normally until the receipt of which is coming from some stream containing a specific pattern or property. Other inputs won't trigger the model to act abnormally; thus, the attack will be very silent.

II.3.1.3. *Challenges for Attackers:*

While there are some weak points associated with federated learning, there still remain several hurdles before data-poisoning attackers [67]:

Influence Restriction: As alters the global model with updates from honest clients, no worthwhile value gets added by a single adversary's contribution in federated learning with many participants. Instead, one needs to come up with some clever techniques to create meaningful effects.

Risk of Detection: If the effect is huge and impacts the behavior of the model, other participants may find it out, either by an anomaly detection system or by the behavior change of the model, which could lead to the discovery of the adversary's action.

Defensive Aggregation: Techniques for robust aggregation can include the following items: trimmed mean, median-based aggregation, or Krum. All these can reduce the diversity of outlier updates and thus weaken the effect of measures against poisoning attempts.

Model Validation: This means that some federated systems use validation on data held out to detect performance degradation and thus find poisoning attempts.

II.3.2. **Model Poisoning:**

Model poisoning is a more direct and possibly deadlier threat than data poisoning attacks in a federated learning environment. In this attack modality, malicious participants do not just tamper with the training data themselves; they also manipulate the model updates directly, resulting in a more accurately targeted effect on the global model [29].

II.3.2.1. ***Mechanisms of Model Poisoning:***

Model poisoning in federated learning can be initiated by various advanced techniques [71]:

Parameter Manipulation: The adversary changes its local model parameters before updating the server, which avoids the entire training process in a most controllable way [30].

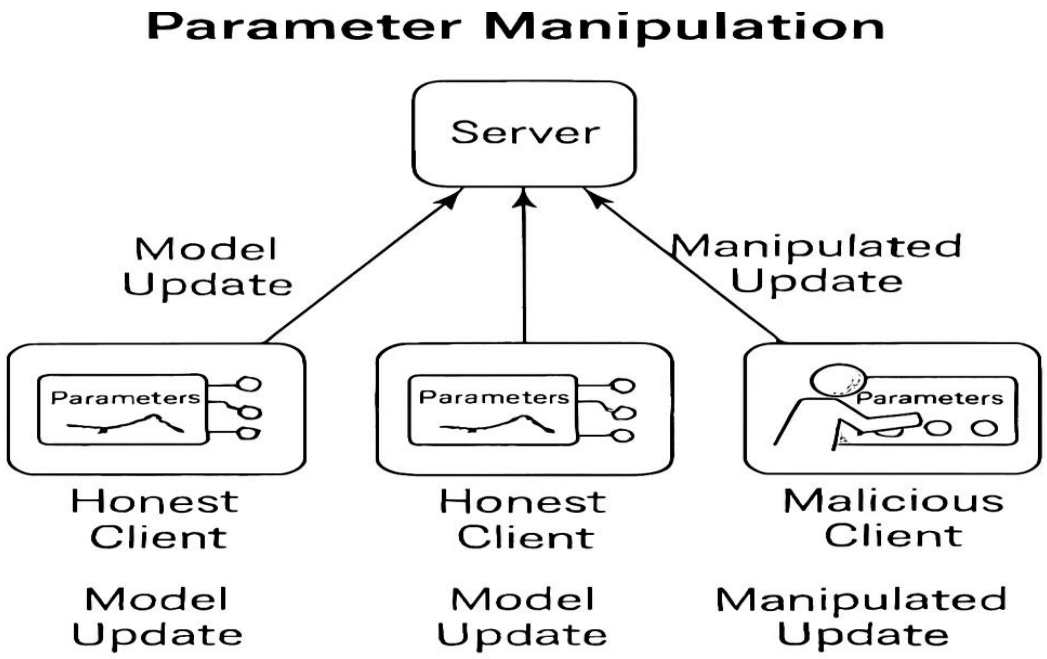


Figure II-3:diagram illustrates Parameter Manipulation Attack

Gradient Manipulation: Instead of modifying parameters in this way, the adversary will change the gradients calculated during training. This can either be used for attacking

purposes or, if camouflaged properly, can save a few detection methods focusing only on the model behavior [30].

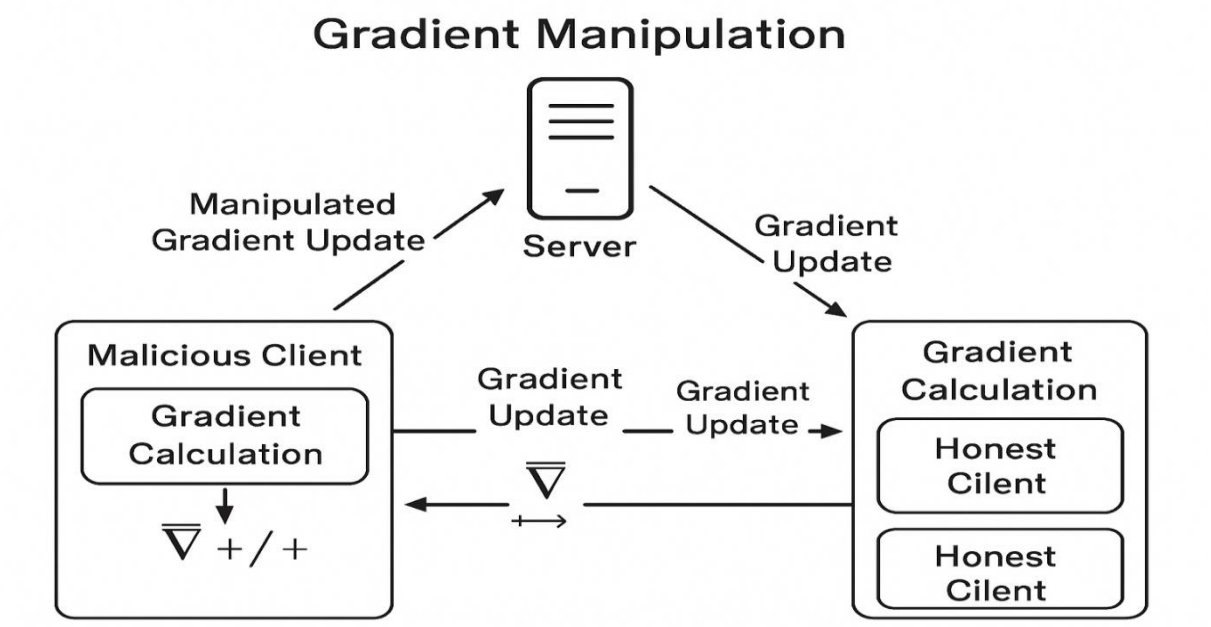


Figure II-4:diagram illustrates Gradient Manipulation Attack

Scaling Attacks: an attacker can amplify his model update by multiplying it by a sufficiently large constant in order to drown out honest user contributions to the global model.[31]

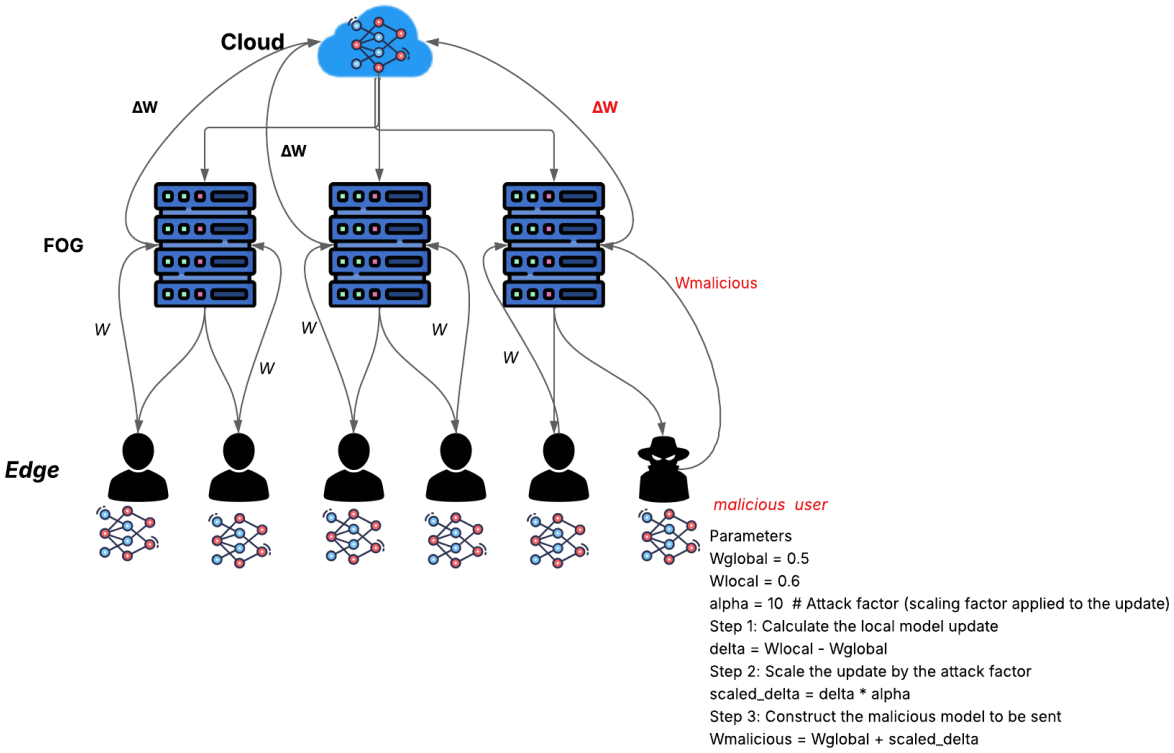


Figure II-5: Scaling Attack Schematic

Deliberate Deviation: carefully constructs updates that change a particular direction of the model which benefits the adversary's objectives possibly over the course of several rounds of training, the effect can accumulate.

Model Replacement: However, in its most extreme form, the attacker would just aim to change the whole global model in favor of the malicious one, which normally would require at least a majority of participants compromised or some flaws in the aggregation mechanism.

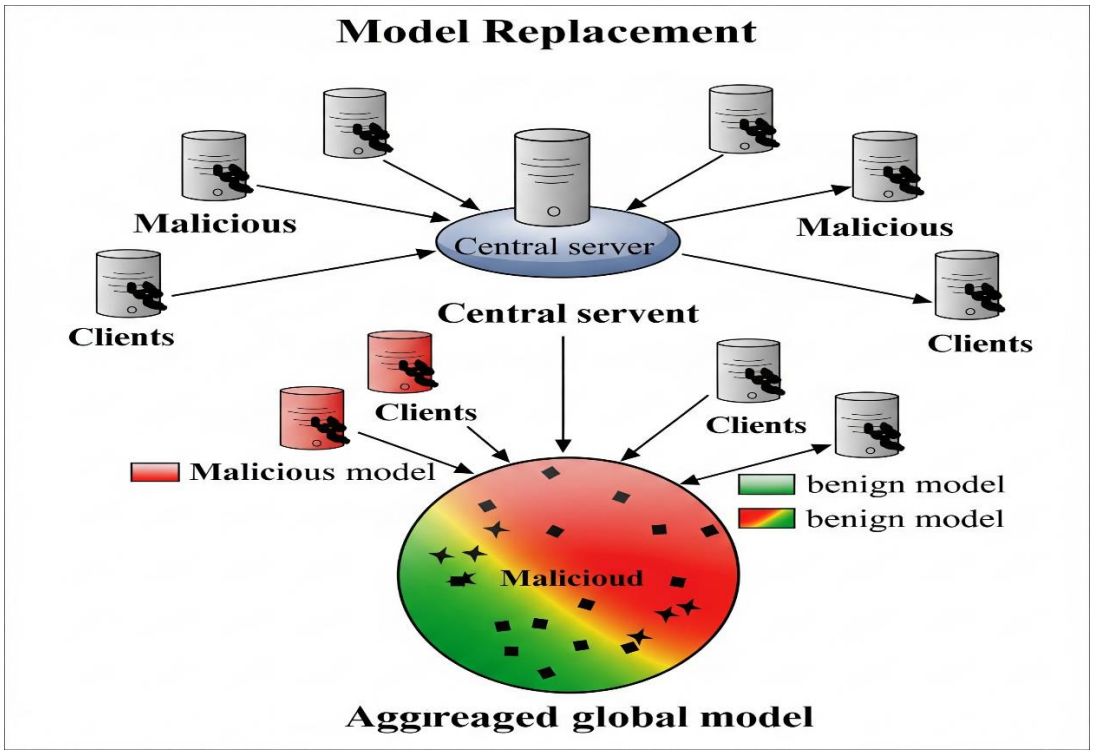


Figure II-6: Model Replacement Attack Schematic

II.3.2.2. *Poisoning Objectives:*

Different objectives are pursued by model poisoning attacks, each causing different implications [72]:

Performance Degradation: Lowering the accuracy of the model overall, hence devaluing it generally for all participants. This can stem from either competitive advantage or simply denial of service.

Targeted Misclassification: Causing profiling misclassifications on select inputs whereby performance on all others remains acceptable. This method is usually much more stealthy than the general performance degradation and is therefore used for targeted attacks.

Backdoor Injection: Similar to data poisoning backdoors but enacted at the model level, creating hidden functionality triggered by specific inputs and behaving normally otherwise [32].

Convergence Disruption: Disallowing the federated learning process from converging to a stable model so all participants are denied service for a long time.

Model Bias Amplification: Manipulating the model to amplify biases already present or inject new ones, raising models' ethical or fairness implications during deployment.

II.3.2.3. *Comparison Between Model Poisoning and Data Poisoning:*

Figure II-7 presents a clear and structured comparison between model poisoning attacks and data poisoning attacks. The figure is divided into two main parts:

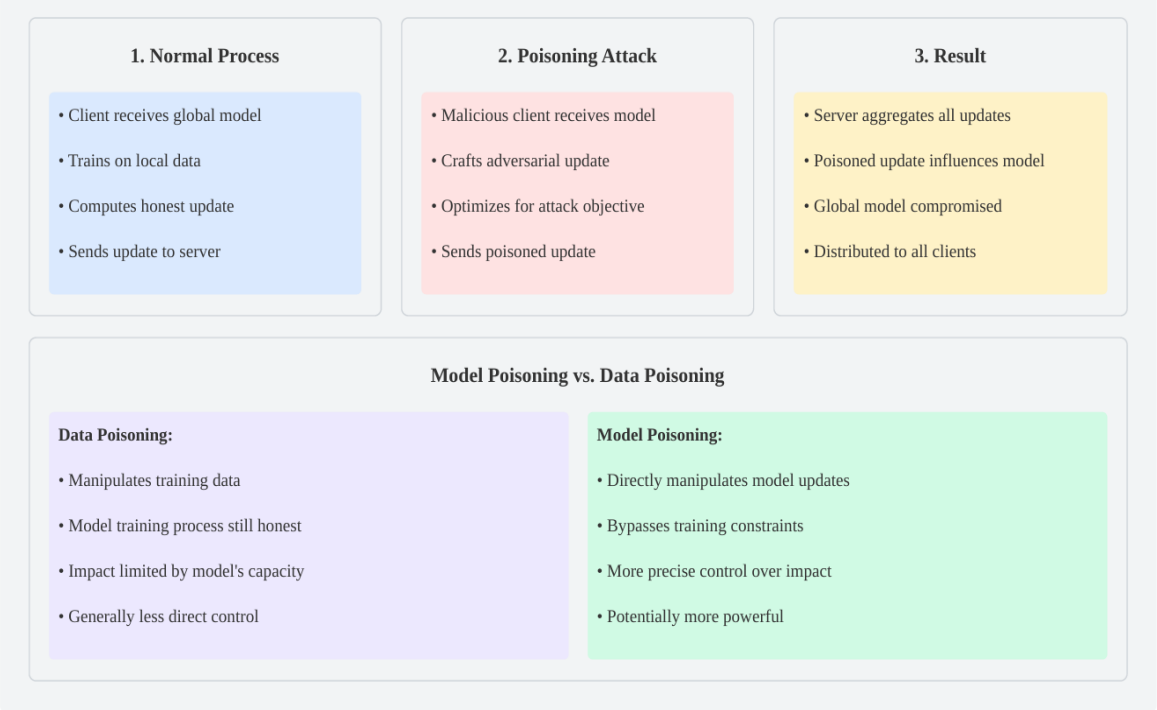


Figure II-7: Model Poisoning Attack Process and Comparison with Data Poisoning

This figure serves as an essential visual tool for understanding the fundamental differences between these two types of attacks, and highlights why model poisoning poses a particularly significant threat in federated learning systems.

In federated learning, model poisoning is a complex and probably very effective attack methodology that lends itself to complex defenses. The pressing need for combating such attacks to ensure the dependability and credibility of federated learning systems in the security-critical applications has already been manifested through extensive research in this area.

II.4. Byzantine Attacks:

Byzantine attacks are a class of attacks wherein the malicious participants, called Byzantine nodes, can act arbitrarily and in potentially contradictory ways. In the context of federated learning, these attacks mainly aim at disrupting the learning process and pre-venting convergence of the global model towards an optimal solution [33], [34].

II.4.1. **Characteristics of Byzantine Attacks:**

"Byzantine" refers to the Byzantine Generals' problem, a classic metaphor in distributed computing that captures the difficulty in coming to a consensus in a system where certain participants may be faulty or malicious.

In federated learning, a byzantine client can exhibit any of the following behaviors:

The difference between Byzantine attacks and poisoning attacks is that Byzantine attacks are generally much more general and potentially even less predictable. Poisoning attacks usually follow a particular strategy towards some target objective, while Byzantine attacks include behaviors that turn up arbitrarily not following any predictable model.

II.4.2. **Common Byzantine Attack Strategies:**

Byzantine attack interruptions can occur almost at random. However, the pertinent literature has identified and examined a number of approaches, and steps have been done to offer solutions [35].

Sign-Flipping Attack: The attacker computes legitimate model updates but then reverses the sign of the updates, effectively pushing the model in the opposite direction of improvement. This simple attack can be surprisingly effective, especially with coordination among attackers.

Gaussian Attack: The attacker sends random updates drawn from a Gaussian distribution to noise the training process such that it slows down convergence or prevents it from happening altogether.

A Little Is Enough Attack: So-called after the paper that introduced it, this sophisticated attack computes a small perturbation that maximizes the loss on a target task over the global model while being hard to detect due to its small magnitude.

Inner Product Manipulation: The attacker crafts updates that manipulate the inner product with the true gradient so as to guide the model further in the wrong direction.

Alternating Minimization Attack: In this type of attack, the attacker alternates between sending honest updates and malicious ones, with the idea being to make his own behavior pattern undetectable through consistency checking.

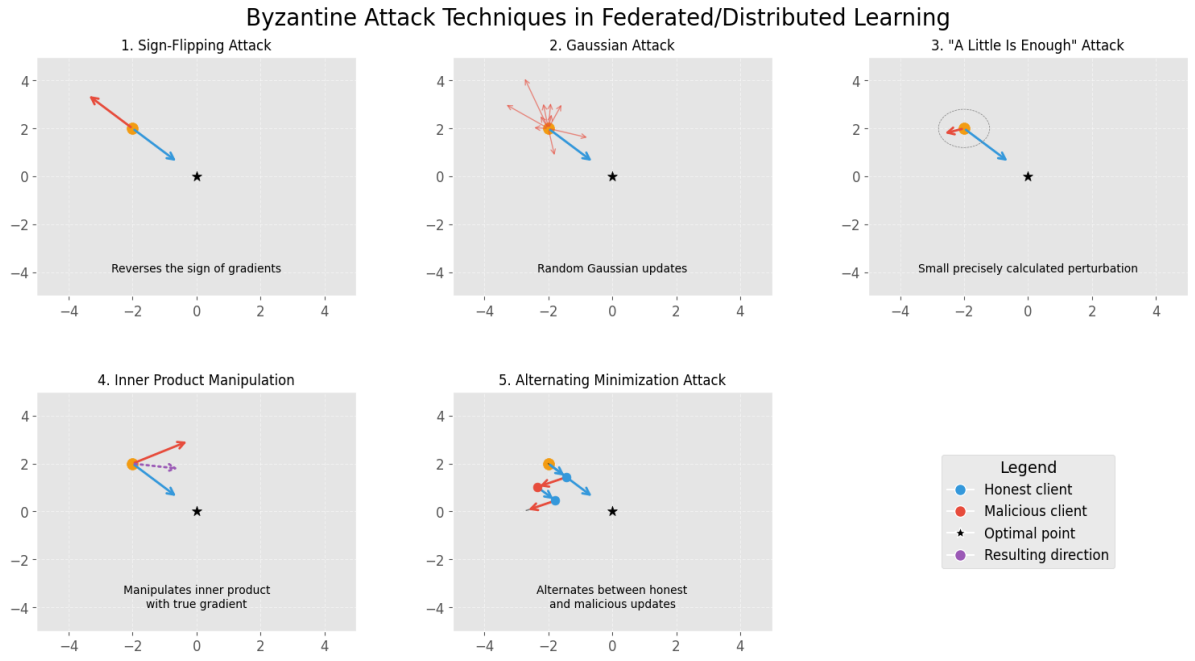


Figure II-8:Visualization illustrate five byzantine attack techniques in federated learning

II.4.3. **Impact of Byzantine Attacks:**

Disruptive effects of Byzantine phenomena on federated learning could be in-many-fold:

Convergence Failure: Indeed, the main aim of Byzantine attacks is exactly to prevent the model from converging with any useful state, thereby denying the outcome to all participants in the federated learning system.

Performance Degradation of the Desired Model: Even if convergence happens, a Byzantine attack can substantially lower the accuracy or performance of a model to start working against its intended purpose.

Increased Convergence Time: Even if the Byzantine attacks do not completely hinder convergence, they can greatly delay the process of convergence, introducing costs in terms of computation and delaying the time at which an effective model is available.

Direction Bias: There are some Byzantine attacks, which seek to make the model biased in certain preempted directions, thereby weakening the model or introducing behaviors in the model considered beneficial to the attacker.

II.4.4. **Challenges in Byzantine-Robust Federated Learning:**

There is still a great deal of challenges to be tackled along with developments made in designing Byzantine-robust federated learning systems:

Non-IID data: Most Byzantine-robust methods are designed under the assumption that data across the clients is identically distributed. In practice, while federated learning operates under heterogeneous data distributions, it makes distinguishing between legitimate distribution shift and Byzantine behavior much harder.

Computational Overheads: These methods for Byzantine-robust aggregations introduce copious computational overhead, which could render them impractical for large-scale or resource-constrained federated learning settings.

Adaptive Attackers: The set of adversaries changes as effective defense mechanisms continue to evolve-a fast-moving arms race. Only a few methods today offer a guarantee against adaptive Byzantine attackers who can observe and retaliate against the developed defensive techniques.

Trade-offs with Privacy: Some Byzantine mechanisms of protection can clash with privacy preservation, putting tension under system design that must be properly balanced.

Byzantine attacks form a key set of challenges in federated learning, testing the limits of distributed trust in collaborative model training. While a good number of Byzantine-robust methods have made their mark, the inherent difficulty of having open participation while ensuring security is one major consideration in federated learning system design [34].

II.5. Backdoor Attacks:

A backdoor attack is one of the most serious threats to federated learning systems, where an adversary schemes to plant hidden functionalities in the trained model that will activate on specific inputs and work as normal otherwise. This undercover operation makes it rather difficult to detect backdoor attacks using regular monitoring means [36].

II.5.1. **Backdoor Attack Fundamentals:**

There are some characteristics that make backdoor attacks really stand out in federated learning.

Trigger Mechanism: Backdoors are created with a specific trigger in mind, which can be a pattern, feature, or condition activating the sabotage act. These include visual patterns against an image, language text sequences, and unique feature combinations in structured data [37].

Stealthiness: Unlike generic poisoning attacks, which are harmless to both the normal data and the poisoned data, backdoor attacks do not disturb the model performance on benign inputs without the trigger, thus eluding detection via conventional evaluation metrics.

Persistence: Well-designed ones can remain active through several federated learning rounds and even continue in its operation after the adversary ceases to participate.

Specificity: Backdoors often target specific misclassifications or behaviors rather than some general model disruption, granting fine-grain control over the attack's effectiveness.

II.5.2. **Implementation Approaches in Federated Learning:**

Various methodologies can exist through which backdoor attacks can be perpetrated in federated learning:

Data poisoning backdoor: The attacker modifies a fraction of their local training data to contain the trigger pattern and associate it with the target label. Their local model establishes the backdoor association learning on this poisoned history.

Model poisoning backdoors: In this setup, the adversary is injecting backdoor functionality either directly by acting upon parameters of local model weights or helping data poisoning to build stronger backdoors among the other data poisoning methods, giving more staying power to the injection.

Distributed backdoors: Different adversaries collaborate to execute different components of a backdoor across multiple rounds or clients so that the collective attack is composed of several disjointed updates that appear together to be a coordinated attack without suspecting a single update.

Adaptive backdoors: The adversary adapts their strategy to implement the backdoor based on the behavior of the model in previously observed rounds to incrementally lead the model to implement the desired backdoor functionality yet remain hidden from detection barriers.

II.5.3. **Backdoor Attack Categories:**

Backdoor attacks in federating learning can be grouped based on many dimensions [88]:

II.5.3.1. ***Based on the Trigger Visibility:***

Visible Backdoors: It uses perceptible triggers, like placing small pattern in corner of an image. Little harder to implement, but they are easier to detect with appropriate approaches to model inspection.

Invisible Backdoors: Using triggers that are imperceptible to human observers, such as very subtle pixel manipulations or feature perturbations, which makes detection by the naked eye practically impossible.

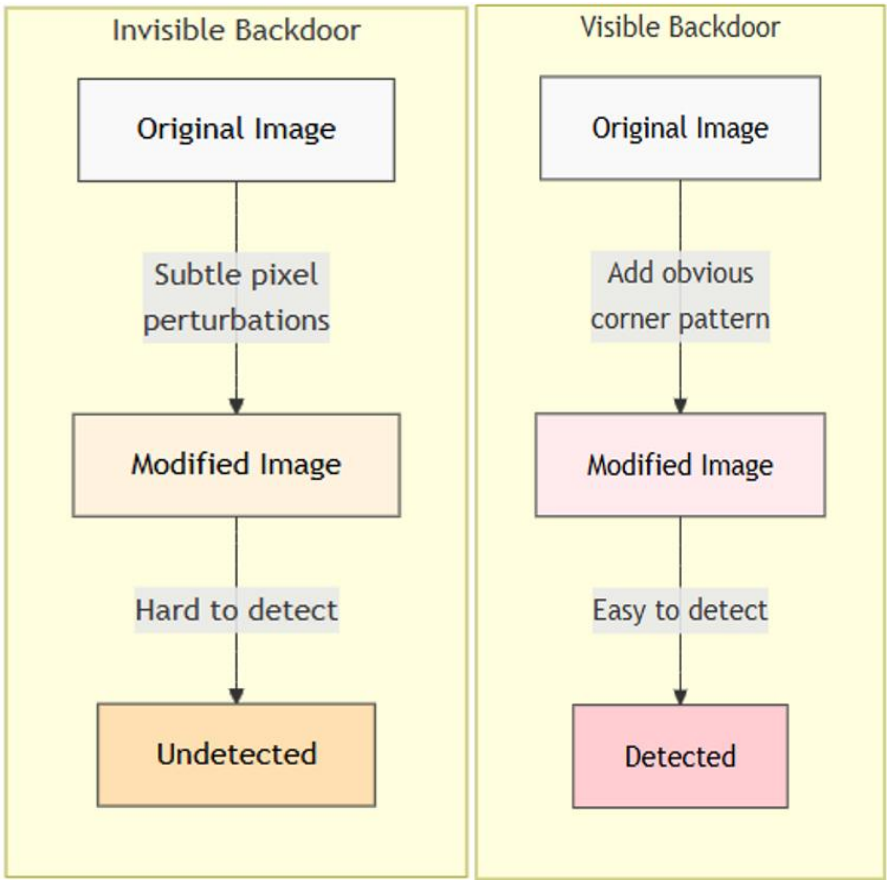


Figure II-9:Trigger Visibility Comparison

II.5.3.2. ***Based on Trigger Scope:***

Input-specific Backdoor: Refers to the backdoor that would just misclassify a specific input or input classes to which the backdoor is applied and would misclassify only that kind of input if the backdoor trigger is present.

Universal Backdoors: The backdoor applies to any input whenever the trigger is present, thus providing larger latitude to an adversary to control the behavior of the model.

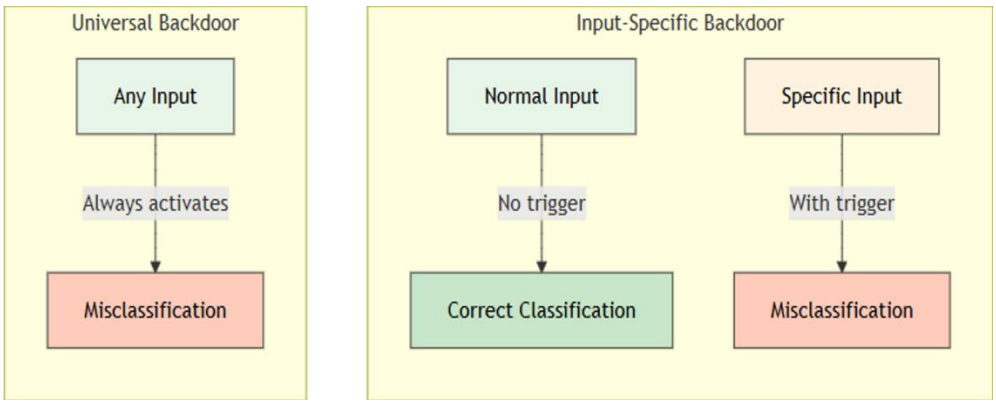


Figure II-10:Trigger Scope comparison

II.5.3.3. ***Based on the Target Specificity:***

Targeted Backdoors: It causes a misclassification into a certain target class chosen by the attacker, thus enabling the fine-tuning of the compromised behavior of the model.

Untargeted Backdoors: causes inputs with the trigger to be misclassified to any wrong class, focusing on general disruption rather than specific outcomes.

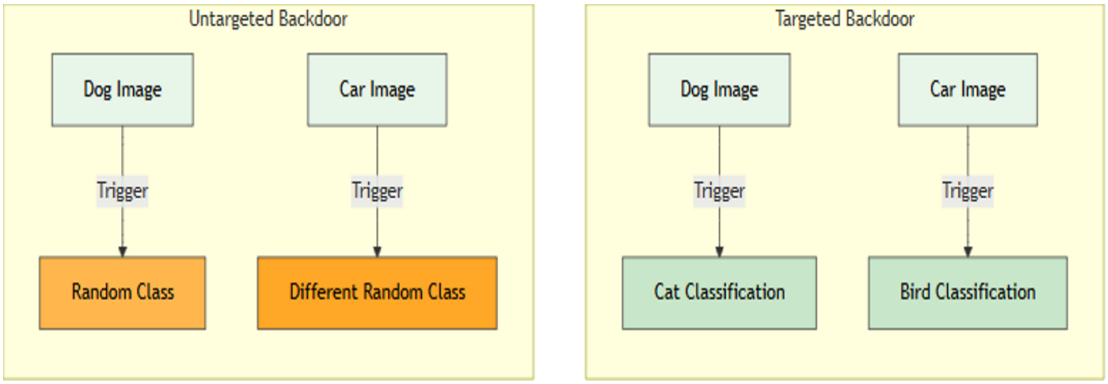


Figure II-11:Target Specificity comparison

II.6. Sybil Attacks:

A Sybil attack is a dangerous security hazard for every decentralized system based on reputation and voting schemes. The circumstances behind dissociative personality disorder's name and Sybil's formation are traced back to 2002 when John R. Douceur coined the name for a security threat; here, the attacker creates several fake identities called "Sybils" to manipulate the network. These fictitious identities are then able to skew results, interrupt communications, and damage reputations.

The susceptibility of an individual system to a Sybil attack is affected by other factors, including identity forgeability, reputation reliance, and acceptance of unverified participants. Sybil attacks have been observed in peer-to-peer networks, early blockchain systems, social sites, and IoT ecosystems. This means that a consequence of such an attack can lead to corrupting data sharing, double-spending in blockchains, or sowing disinformation [38], [39].

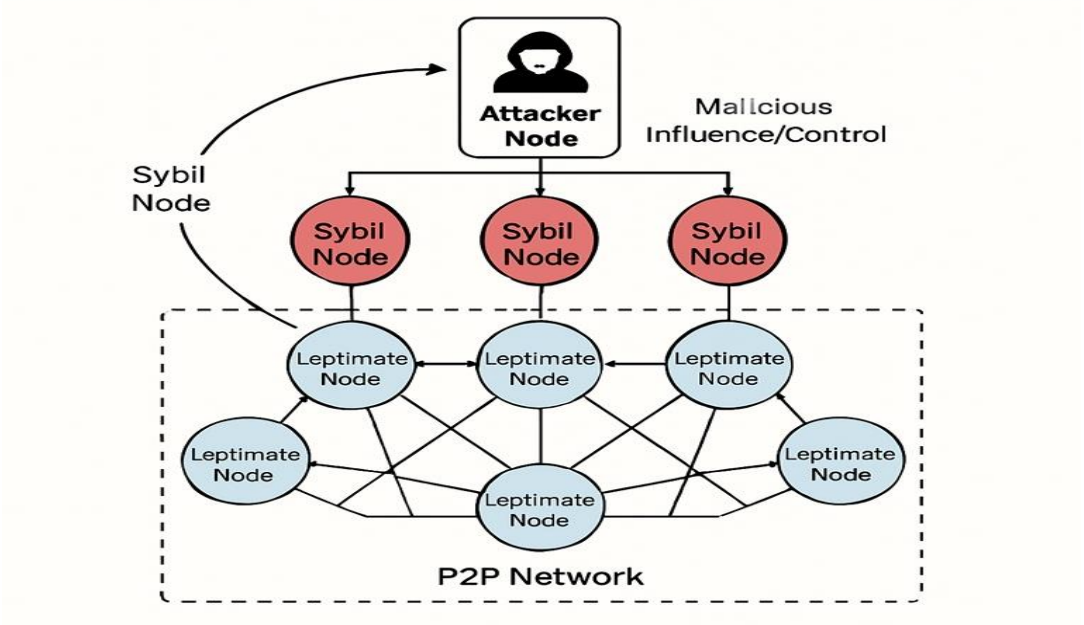


Figure II-12:A one attacker node controls many Sybil node to affect a P2P Network

- There are a couple of distinct types of attacks on the system:
  - A direct Sybil attack: The use of fake accounts directly to interact with real users (that is, posting fake reviews).
  - An indirect Sybil attack: Here, the trusted intermediate nodes are brought under the influence of the Sybil nodes that spread their malicious effects indirectly (this can be, for example, manipulating IoT devices through a compromised router).

II.6.1. **Challenges for Sybil Attackers:**

The fact that Sybil attacks considerably endanger decentralized systems poses several threats and versatility to an attacker owing to the imposed defense mechanisms and the very nature of distributed networks:

Costs of Identity Creation: Creating and maintaining a large number of believable fake identities could be an expensive endeavor. Systems that impose identity verification (e.g., phone or credit card checks) or economic barriers such as Proof-of-Work or Proof-of-Stake schemes render large-scale Sybil attacks economically unfeasible.

Validation Slice: Attackers must ensure that their fake identities will pass the system's validations, such as CAPTCHAs, computational puzzles, or even Proof-of-Personhood protocols. Those are hard to automate at a large scale and require larger amounts of effort and resources.

Detection Through Network Analysis: Defensive systems perform social graph analysis (e.g., SybilGuard, SybilLimit) to detect suspicious connection patterns. Attackers need to

carefully design their fake networks such that they do not come under detection, which itself is a major technical challenge.

The cost of resources: The maintenance of a multitude of active Sybil identities will eat away at computer power, bandwidth, and storage, making the attacker's job even more expensive and complex.

Behavioral Consistency: Fake identities ought to conduct themselves realistically and in a consistent fashion over time, so as to avoid growing a suspicion or being flagged for unusual behavior.

Risk of Exposure: Detection or failed validation attempts may lead to banning some of the Sybil identities, wasting the attacker's resources and probably exposing their technique.

II.6.2. **Common Sybil Attack Patterns and Scenarios:**

Sybil attacks are the most dynamic attacks since they can modify themselves according to the system and target objectives. The attackers use a number of fake identities to manipulate, interrupt or get unfair advantages. Some of these include:

- **Voting Manipulation:** Such identities Sybil use in systems such as DAOs or blockchain governance to increase voting power proportionately for dependent decision-making.
- **Reputation System Subversion:** Fake accounts are used for generating a false reputation (up or down) for products, services, or users (e.g., fake reviews).
- **51% Attacks in Blockchain:** By many nodes or stakes, attackers will manipulate the transaction validation, permit the double spending or will destroy consensus.
- Disruption or Censorship: Sybil nodes can block or delay message propagation, effectively silencing users or launching denial-of-service (DoS) attacks.
- **Deanonymization (as in Tor):** those with control over many ingress/egress relays can track user traffic, thus compromising anonymity, at times using a combination of other techniques such as SSL stripping.
- **Routing Attacks:** In a P2P or ad-hoc scenario, the presence of a Sybil node redirects or drops in most cases, traffic by providing erratic routing information.
- **Storage/content Poisoning:** Fake nodes will give unsolicited and deceptive data to distributed commons such as BitTorrent DHTs and reduce reliability. Spread malicious software.
- **Pseudo-impersonation in Airdrops/Giveaways:** Fake identities created in the crypto ecosystem to fraudulently reap the benefits accrued to unique users.

In fact, Sybil attacks can aim at almost all decentralized systems, customizing the tactics according to specific weaknesses in identity, trust, and consensus mechanisms.

II.7. Other Network Security Threats:

II.7.1. **Denial-of-Service (DoS) Attacks:**

In simple words, it is a method to deny every legitimate user of accessing the service in a machine or a network resource. The attack usually functions by sending in excess traffic or some vulnerability that makes a system crash.

Denials-of-Service attacks manifest themselves in the following manners-from affecting performance to a complete outage of service-website, financial systems, and, most critically, infrastructures. In most cases, the reasons include revenge, extortion, hacktivism, or simply halting services.

More sophisticatedly distributed denial-of-service (DDoS) attacks employ a multitude of compromised devices-most of which constitute a botnet-to endlessly barrage the target from many corners of the world. That renders DDoS attacks incredibly more difficult than classic DoS [40], [41].

- There are many different types of DDoS attacks:
  - Volumetric attacks-e.g. UDP floods-use consumption of bandwidth.
  - Protocol attacks (e.g., SYN floods)-attack mechanisms based on weaknesses in network protocols.
  - On application-layer attacks (e.g. HTTP floods)-overlap with just some particular services with requests that are, on the surface, legitimate looking.

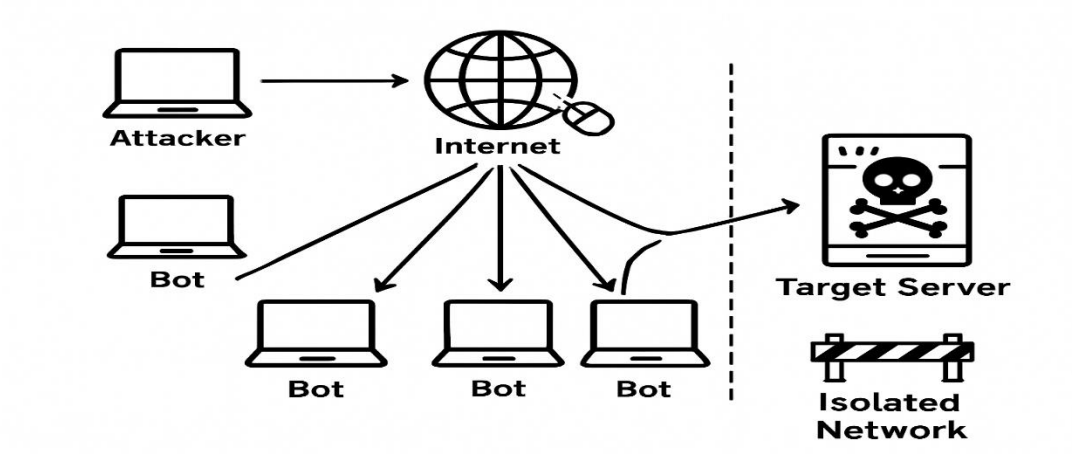


Figure II-13:A coordinator directs a botnet to flood a target server with traffic, causing congestion and blocking legitimate users

II.7.2. **Man-in-the-Middle (MitM) Attacks:**

Man-in-the-Middle (MitM) attack is characterized by the silent interception and possible alteration of communication between two parties who think they have directly and securely interacted with each other. The attacker acts as a hidden relay impersonating either side in order to access sensitive information including login credentials, logs of financial transactions, or confidential communications [42], [43].

MitM attacks will generally be described in relation to:

- Phase 1 - Interception: traffic is somehow routed through the attacker's system:
  - rogue Wi-Fi hotspots
  - IP spoofing
  - ARP spoofing (local manipulation of address mappings)
  - DNS spoofing (hijacking users to fake websites)
- Phase 2 - Possible Decryption: If the communication is encrypted, attackers will use some of the following ways:
  - HTTPS spoofing (faking certificates)
  - SSL/TLS exploits such as SSL BEAST
  - session hijacking
  - SSL stripping: use HTTP instead of HTTPS

These attacks usually target users who use financial services, a SaaS platform, and other authenticated systems.

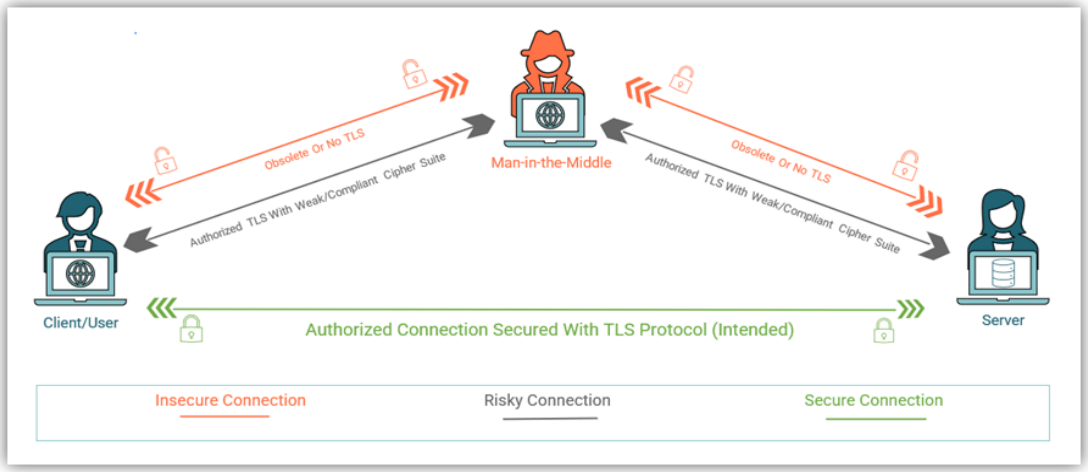


Figure II-14:A visual representation of a man in the middle attack and the difference between a secure (HTTPS) and insecure (HTTP) risky connection

II.7.3. **Replay Attacks:**

A Replay attack is a type of network attack in which an illegitimate user captures and later transmits a valid piece of data like authentication credentials or transaction requests in order to impersonate a user or again perform unauthorized actions to the user. In replay attacks, data can passively capture or record for later misuse, like harbored and reused by the hash password, where real-time manipulation does not exist such as in man-in-the-middle attacks.

Lines against replay attacks require that recycled messages be not accepted by the systems. Among the common defenses are:

- **Session IDs or Tokens:** unique values assigned to each session preventing cross-session use.
- **Nonces:** one-time-use numbers embedded in messages ensuring freshness.
- **Timestamping:** messages are only accepted within valid time ranges (with synchronized clocks).
- **One-Time Passwords (OTPs):** passwords that have a time limit for expiration after a single use.
- **Challenge-response protocols:** like CHAP dynamic challenge values used for authentication.
- **Secure protocols:** such as Kerberos which has time-limited tickets.

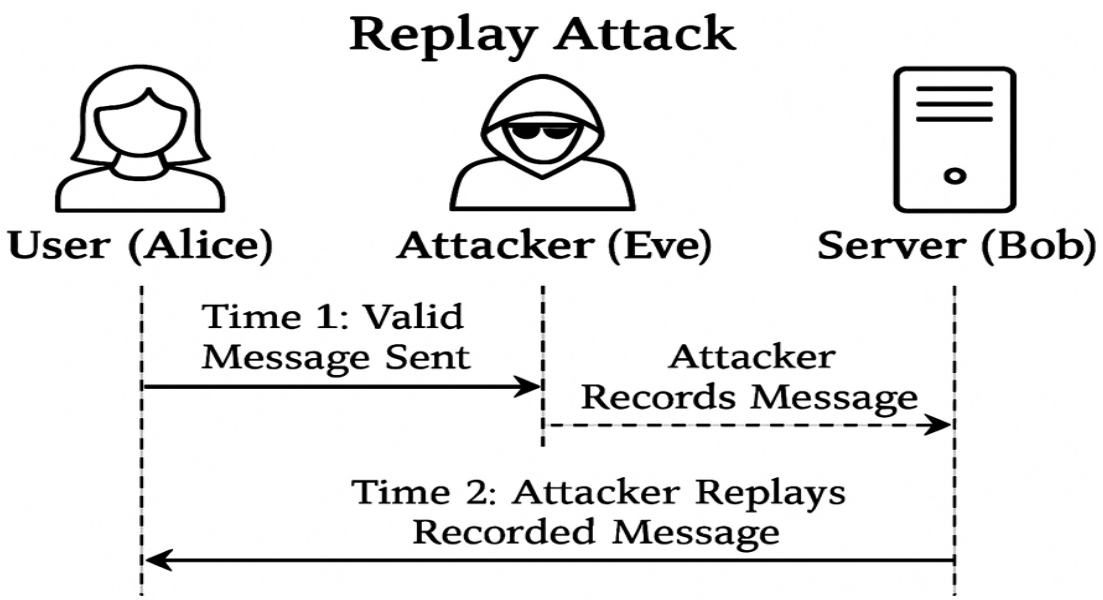


Figure II-15: A visual representation of attacker records a valid message sent at Time 1 and replays it at Time 2 to impersonate the user or repeat an action.

Having thoroughly detailed the various adversarial threats that plague Federated Learning, including poisoning, Byzantine, Sybil, and network-level attacks, it becomes crucial to consider the inherent defensive capabilities of different aggregation strategies. While a more in-depth analysis of these defense mechanisms is comprehensively presented in **Chapter III** ,**Table II-1** provides an initial comparative overview of prominent aggregation strategies. This table highlights their varying degrees of robustness against the aforementioned attack types, their support for non-IID data, and their associated computational overheads. This overview serves to underscore the foundational role of robust aggregation in securing FL systems against the very vulnerabilities discussed in this chapter.

Table II-1:Comparison of Aggregation Strategies in Federated Learning

Countermeasure	Mitigated Attack Type	Key Strength	Key Limitation	Overhead
Local Data Filtering	Data Poisoning	Removes anomalous or mislabeled samples before training [44]	Bypassed by advanced feature-level attacks; limited under non-IID	Low
Provable Data Provenance	Data, Hybrid	Ensures data integrity using blockchain or TEE [45]	Requires hardware support or blockchain integration; scalability concerns	Medium

Differential Privacy	Data, Model, Hybrid	Bounds per-client influence via noise injection [46]	Utility-privacy tradeoff; sensitive to non-IID data	Medium
Robust Aggregation (Krum/Multi-Krum)	Model, Hybrid	Filters outlier gradients; resists Byzantine updates [47]	Vulnerable to similarity-based attacks (Sine) [47]	Medium-High
Trimmed Mean / Median	Model, Hybrid	Robust against sparse malicious updates [47]	Fails under dense Sybil or adaptive attacks	Low-Medium
Adaptive Clipping	Model, Hybrid	Suppresses large gradient deviations [48]	Attackers can align gradients to evade detection	Low
Anomaly Detection & Client Screening	Model, Hybrid	Detects suspicious updates using similarity metrics and clustering [44]	False positives in non-IID scenarios; adaptive attacks remain stealthy	Medium
Cluster-Based Filtering	Data, Model, Hybrid	Detects coordinated Sybil attacks using clustering techniques [48]	Sensitive to client heterogeneity; parameter tuning needed	Medium
Joint Data–Gradient Monitoring	Hybrid	Correlates local data and gradients to detect stealthy poisoning [44]	Computationally intensive; feature-level monitoring required	High
Multi-Round Consistency	Model, Hybrid	Flags temporal inconsistencies in updates across rounds [46]	Adaptive attackers can mimic consistency	Medium
Generative Defenses (GANs)	Hybrid	Models clean data distribution to detect anomalies [49]	Needs adversarial training; resource-intensive	High
Secure Multi-Party Computation (MPC)	Model, Hybrid	Aggregates encrypted updates while preserving privacy [46]	Communication overhead; increased latency	High
Trusted Execution Environments (TEE)	Model, Hybrid	Isolates secure computation in hardware enclaves [46]	Requires hardware deployment; scalability limits	Medium
Homomorphic Encryption	Model, Hybrid	Enables encrypted update aggregation [46]	Computationally heavy; slower than plaintext aggregation	Medium-High

Audit Trails	Model, Hybrid	Provides post-hoc accountability for model updates [46]	Not proactive; requires manual investigation	Low

II.8. Taxonomy of Privacy Attacks in Federated Learning:

To have a comprehensive understanding of privacy threats, it should be systematic and organized classification. Through a thorough review, privacy attacks can be classified across different analytical dimensions in the existing literature. The table that follows presents a fine-grained taxonomy of privacy threats that are specific to federated learning systems, putting together the essential findings from current scholarship [50].

Table II-2:Updated Taxonomy of Privacy Attacks in Federated Learning

Classification Axis	Category	Description	Attack impact
Type of Information	Membership Inference[51]	Determine if a specific data record was part of the training set.	Compromises privacy by revealing sensitive individual participation
	Property Inference[52]	Make inferences about training data distribution about general properties or statistics (for instance, ratios of demographic characteristics, the presence of certain features).	Reveals statistical patterns about the training population, potentially exposing sensitive aggregate information.
	Data Reconstruction[50]	Repair (partially or fully) a small fraction of original training data samples by model updates or the final model.	Exposes private or sensitive training data, leading to potential data breaches.
	Model Extraction / Stealing	Create a substitute model that mimics the functionality of the target model, potentially stealing intellectual	Leads to intellectual property theft and loss of competitive advantage.

		property.	
Malicious Actor	Malicious Server (Aggregator)	The unreliable server tries to derive information from messages of the client between itself and the client, and hence block synchronous behavior to observe it critically, with an implicit hint to the "Honest-But-Curious" mentality.	High risk: central entity can compromise all connected clients' data.
	Malicious Client(s)	One or more of the participating clients might try to infer the information about other clients' data or poison the model so as to create leakage.	Threatens both data privacy and model integrity within collaborative frameworks.
	External Eavesdropper	An outside attacker intercepts communication or queries the final deployed model.	May lead to exposure of model behavior or inference on training data.
Adversary Knowledge	White-box	Attacker has full access to the model (architecture, parameters, gradients, training algorithm).	Maximum impact due to full visibility into the system's internals.
	Gray-box	Attacker has partial knowledge-meaning not complete model architecture, but still missing updates.	Moderate impact; allows partial reconstruction or inference.
	Black-box	Adversary as such have direct contact with the model through an input/output interface like with the final resulting model that	Lower technical complexity but still enables membership or property inference.

		has been deployed, possibly querying it.	
Attack Phase	During Training	Attacks exploit the information exchanged during the iterative FL training process (e.g., gradients, model updates).	Exposes sensitive information from training data through gradient leakage.
	Post-Training (Inference)	Full deployment of the global model occurs after the end of the training process. The attacks locate their targets in preemptive post-training events.	Enables membership inference or property inference attacks.
Attack Vector	Gradient Analysis	Directly analyzing gradients or model updates sent by clients.	Similar to during-training attacks; risks re-emerge in deployment phase.
	Model Output Analysis	Analyzing the predictions and confidence scores of the local or global model.	Allows black-box attacks such as membership inference and model stealing.
	Generative Models (e.g., GANs)[53]	Another possible view could involve using generative models to synthesize data so as to match the cultural specifications learned by the FL model thus leaking training data.	Reconstructs training data or mimics model behavior without direct access.
	Side Channels	Exposing messages that are compromising timings or communication patterns based on system resources.	Leverages indirect signals to breach privacy or extract model knowledge.

II.9. Membership Inference Attacks:

MIA attacks attempt to ensure membership of a particular data point used in the training of a target model. Such attacks pose a serious threat to data privacy- particularly when the data in consideration are sensitive in nature, such as medical records or financial information [51], [53], [54].

II.9.1. **Attack Principle:**

The same circuitry was introduced in 2017 by Shokri et al. in their treatment of machine learning and concerns the observation that models behave differently on data they have been trained with versus on unseen data. Such differences include high prediction confidence for members or very subtle changes to the output distributions. The attackers often train so-called shadow models on data sets with known membership in relation to the target data point under consideration. Inference on the attack model (the meta-classifier) can thus be made by comparing the outputs of the target model on the data point in question with the outputs of the shadow models and inferring the respective likelihood of membership (refer to Figure).

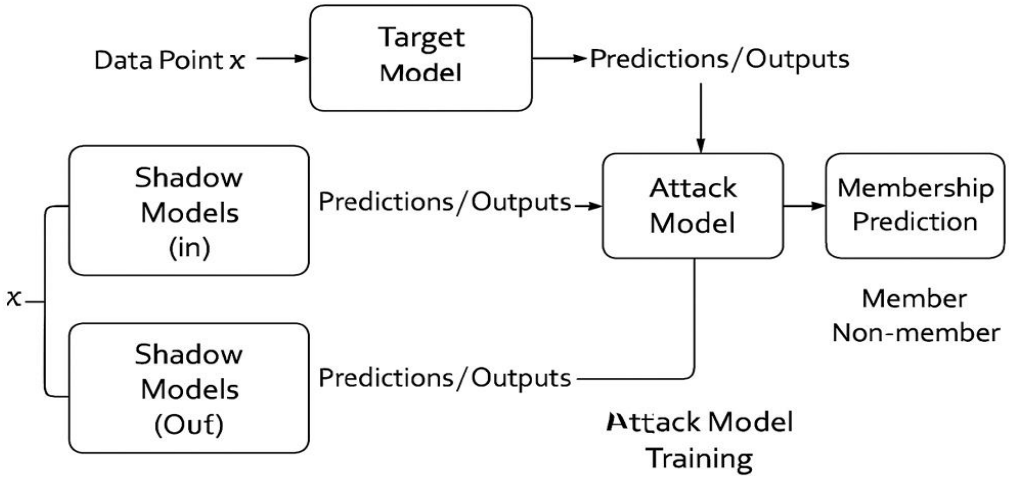


Figure II-16:Illustration of a Membership Inference Attack using shadow models

II.9.2. **Consequences:**

The successful carrying out of a MIA may lead to the revelation of sensitive information (e.g., inferences on medical conditions), de-anonymization, user profiling without consent, and infringements on data privacy regulations such as the General Data Protection Regulation.

II.10. Property Inference Attacks (PIA):

Property Inference Attacks (PIA) tries, in some sense, to extract general characteristics or statistical properties of the training dataset, rather than any specific point in the data. For example, an attacker could try to infer a certain demographic group's estimate or whether a certain feature distribution exists in the overall training data [52], [55].

II.10.1. **Attack Principle:**

Just like MIA or Membership Inference Attack, PIA or Property Inference Attack also commonly makes use of shadow models. Similar to the shadow model approach, the adversary essentially trains shadow models on auxiliary data either with or without the specific property of interest. Thereafter, comparison of the behavior of the target model (for example, its output, weights, or gradients) with that of the shadow models is done such that a meta-classifier can be trained to determine whether or not the target model has been exposed to data containing that property (see Figure II.17). The paper by Ganju et al. (2018) discussed some of the main difficulties in applying PIA to deep neural networks, and proposed neuron sorting and deep-set-based representations to deal with permutation invariance across layers in the model. On additional levels, they also extended the study of PIA to FL environments, showing how an enemy would be able to deduce private properties from shared gradients which do even relate to the main learning task in no way.

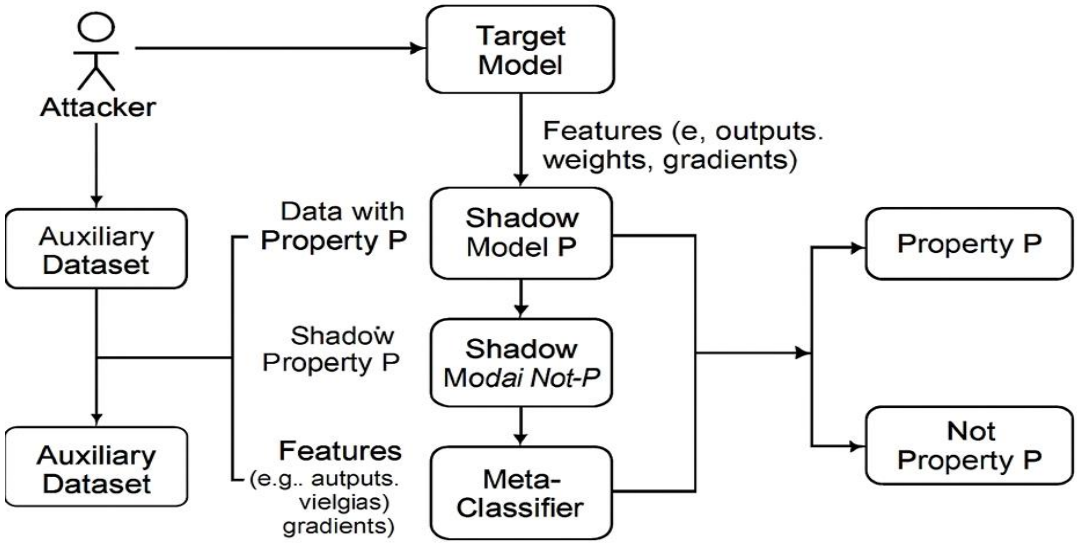


Figure II-17: Visual explanation of a Property Inference Attack, with an attacker training shadow models corresponding to datasets with and without the target property,

II.10.2. **Vulnerabilities and Consequences:**

Sensitive aggregate information about user groups may leak through PIAs, thus facilitating discrimination or exposing business-sensitive statistics. Depending on certain data distributions possessed by some participants, properties may be easier to infer in a collaborative setting like FL.

II.11. Data Reconstruction Attacks:

These attacks represent a more severe privacy breach, aiming to reconstruct the actual training data samples used by clients

II.11.1. **Gradient-Based Reconstruction:**

Very recently, studies have shown that gradients in deep learning can maintain information about the training data-the samples used to compute them-with extremely high fidelity. Zhu et al. (2019) proposed a concept called "Deep Leakage from Gradients," showing that under certain conditions, it is possible to almost perfectly reconstruct the original training images from gradient updates. Federated learning basically deals with the aggregation of model updates through multiple clients, yet there remain vulnerabilities when an adversary has access to updates from just a few participants or when the model architecture is prone to this leakage [16] [56].

II.11.2. **GAN-Based Reconstruction:**

Since Generative Adversarial Networks (GANs) may be put to use for data reconstruction attacks, the trick here is that the adversary—be it a client or server—uses the model updates and/or gradients received as supervisory signals to train the GAN (see Figure II.18). The generator within the GAN learns to generate data samples such that their training with these samples yields a model update similar to that of the observed one. This way, the adversary can reconstruct data points that are structurally and statistically similar to the private training data (Hitaj et al., 2017; Wang et al., 2019) [57], [58]**.**

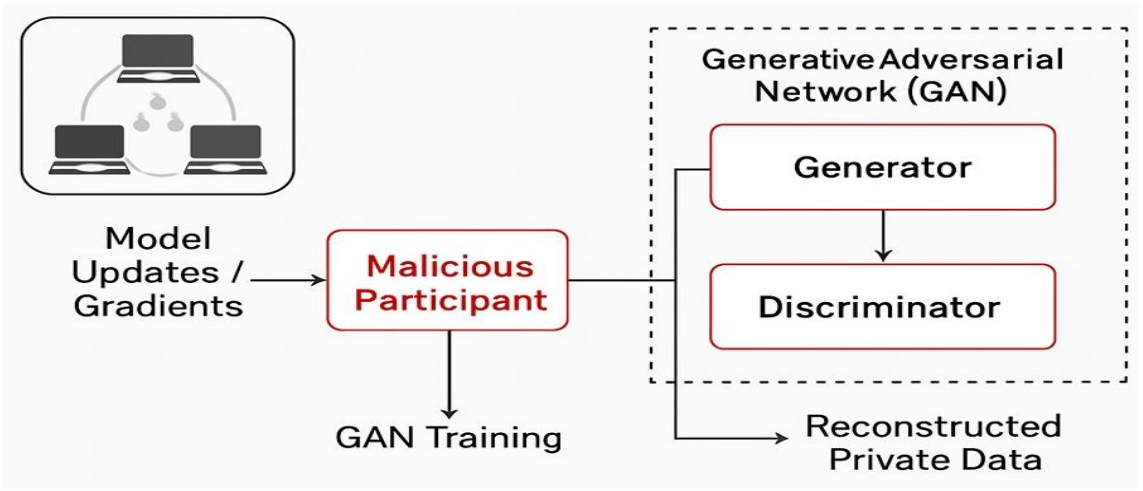


Figure II-18:illustration of a GAN-based reconstruction attack in a federated learning environment. Here, a malicious participant employs model updates or gradients to train a GAN that reconstructs the private training data.

II.11.3. **Consequences:**

A successful reconstruction attack may directly perpetrate the exposure of raw user data and may result in considerable privacy invasions, acts of identity theft, or the breach of highly sensitive information or confidential information.

II.12. Conclusion:

In the end, despite big advantages with respect to safety and confidentiality of data, FL is prone to several attacks that threaten to destroy the integrity of collaboration-framework-even attacks ranging from manipulation of data and poisoning of models to inference attacks and compromised participants.

That being said, the attacks are unique because, although infinitely more difficult than ever, they aim at challenging the integrity, confidentiality, and trusted evaluation of the whole collaborative learning effort. The latter challenge stems from the existence of adversarial participants and insecure communication in FL among client sites and central servers. Gaining insight into such vulnerabilities is vastly instrumental in making decisions toward effectively designing strong defensive measures.

This will lead us to the next chapter, which deals with one of the key ingredients under an FL system, that is, aggregators, which combine the local model updates, their influence on the security of the entire system, and the ways by which they are fortified against such attacks.

Chapter 3: Study of Aggregators in Federated Learning:

III.1. Introduction:

The process responsible for aggregation has become indispensable in the implementation of Federated Learning. The chapter has thus far covered threats and vulnerabilities and proceeds to discuss the operational aspects of aggregation: that is, aggregation refers to the process of uniting the locally trained models from various distributed sites into one common global model. An understanding of the intricacies involved in this process, together with the various methods utilized for aggregation, is a prerequisite for the effective and secure design of privacy-conscious FL systems.

III.2. Role and Importance of Aggregation:

FL essentially capitalizes on decentralized learning rather than conventional machine learning-they lump all the data together in one location where the data is stored. Each device, also called client, trains its own model locally on the data that belongs to the device only. It does not serve the purpose of putting trainers together based on their data but uploading individual device models to a server and combining the parameters they learned. It is then possible during that aggregation stage.

All the clients can very well picture individual researchers doing a segment of the same puzzle. Instead of sharing their raw data, they send in their conclusions. Then aggregation takes those conclusions and assembles one complete image where no one has to actually put on the table their original data sources. According to Qi et al [59], therefore, it can be said that model fusion can also define the framework that enables FL to maintain privacy while training a model collectively.

The importance of aggregation becomes clearer when viewed through several key perspectives:

III.2.1. Collaboration Under Privacy Constraints:

The federated learning paradigm has a central invention, which is to enable joint model training without collecting sensitive information at a broker location. Aggregation is the engine for this privacy-preserving collaboration. By concentrating on the aggregation of model updates changes or parameters and not the data, FL sets up a very basic boundary between the ownership of data and the model improvement [60].

Mathematically, think of a federation having N clients with each client i having a local dataset D_i. In a typical centralized approach, all data would need to be pulled together and put in a central dataset D_i. A exposing all information to a central body. Federated learning, on the other hand, favors data locality. Each client i trains its local model M_i on its private data only, D_i. An aggregation step then combines these local models (or their updates) into a global model M_G without ever needing access to the underlying datasets:

$$M_G = \text{Aggregate}(M_1, M_2, \dots, M_N) \quad \text{(III.1)}$$

It is this process that fundamentally rebalances the trade-off between privacy and utility in machine learning, thereby giving birth to high-quality models developed under severe privacy constraints [61].

III.2.2. Building a Global Model from Heterogeneous Local Insights:

The heterogeneous nature of local data presents a major challenge for aggregation. The datasets Di may vary in size, quality, and statistical distribution. This makes it difficult to extract common patterns necessary for building a high-quality global model. The goal of aggregation is therefore to combine disparate information into a single model capable of generalizing effectively. Key Challenges Solved by Aggregation:

In practice, this involves solving several problems:

Dealing with Different Data (Statistical Heterogeneity): The data is sometimes said to be "non-IID" when it is not evenly spread among the different clients or looks very different from one client to another. Such discrepancies can lead to a large divergence from the local models as they train, thus making aggregation very difficult [60].

Handling Different Systems (System Heterogeneity): Related to the device heterogeneity, as some devices are very powerful and have high-speed Internet connections, while some others are slow or connected intermittently to the Internet. Such system-level differences necessitate possible protocols for achieving synchronization between different devices when the model updates or aggregates [62].

Protecting Privacy: Federated learning keeps raw data stored locally. Still, the aggregation step needs to be designed with maximum caution. Private information must not, through any model updates that can be shared, be unintentionally leaked. The aggregation strategy, furthermore, would have to be secure against malicious participants.

Efficient Communication: In real-world setup scenarios, sending data back and forth between clients and a central server can be extremely slow. Such phenomena are labeled a performance bottleneck. Hence, aggregated methods that reduce both frequency and amount of transmitted data are crucial [59].

III.3. Fundamental Principles of Model Aggregation:

Model aggregation, what is it really in federated learning? In layman's terms, it would mean consolidating several local AI models trained on devices or their updates into a unified global model. It is the machinery for sharing and aggregation of knowledge among all the participants while keeping their data local private [59].

The concepts basic to the working of model aggregation can usually be divided into two varieties, namely:

III.3.1. **Parameter-Based Aggregation:**

This distribution of weights enables the training of a global model based on individual local model states. Parameters represent settings within the local model such as weights and biases, which decide on model behavior. This technique thus compares the guts of local model implementations.

A good example is the Federated Averaging (FedAvg) algorithm set out by McMahan et al [63]. FedAvg computes a weighted average of parameters (i.e., weights and biases) collected from all local models:

$$w_{t+1} = \sum_{i=1}^{N} \frac{n_i}{n} w_{i,t} \quad \text{(III.2)}$$

Here, w_{t+1} refers to the parameters of the global model for the next round $(t + 1)$, $w_{i,t}$ is the local model parameters of client i at the current round (t), n_i represents the number of training samples on client i, and $n = \sum_{i=1}^{N} n_i$ is the overall sample count across all clients.

Parameters-based methods can be further differentiated by their unique weighting schemes; using their employing regularization techniques and mechanisms for handling non-IID data distributions [62]. This common assumption allows the direct averaging of corresponding parameters into the fact that all local models are assumed to be identical in architecture.

From an optimization perspective, parameter-based aggregation can be interpreted as a kind of distributed optimization-whereby the ultimate goal is to minimize a global objective function, which is usually formulated as a weighted average of local objective functions:

$$\min_w F(w) = \sum_{i=1}^{N} \frac{n_i}{n} F_i(w) \quad \text{(III.3)}$$

A loss function calculated with a local dataset determines the local objective for the client. The result of combining these solutions computed locally approximates the solution to this global objective iteratively using parameter-based aggregation methods [64] .

III.3.2. **Output-Based Aggregation:**

In contrast to the methods that follow parameters, rather, output-based aggregation is based on combining the predictions or output from local models and does not touch the internal parameters of these local models. It is helpful when the local models have dissimilar architectures or when the aim is to use an ensemble made of different models [65].

For a given input, each local model M_i generates an output $O_i(x)$ Next, the aggregation step comprises the act of composing these individual outputs into a final prediction $O_G(x)$:

$$\boldsymbol{O_G(x) = \text{Aggregate}\left(O_1(x), O_2(x), \dots, O_N(x)\right)} \quad \text{(III.4)}$$

Usual methods for output assembling are:

- **Majority Voting:** The final prediction is for the class that receives the most votes from the individual local models for mainly classification.
- **Weighted Averaging:** The outputs of the local models are combined by using weights that might be established by several factors, including those influencing model confidence, a model's past performance, or the amount of data used in training.
- **Knowledge Distillation:** A teacher-student framework where the global model (student) replicates the combined outputs of the local models (teachers) [66].

Output-based aggregation does have its advantages: it allows the integration of heterogeneous models; it may reduce communication overhead (if the outputs are smaller than the parameters); and it may increase robustness with respect to certain types of attacks. On the downside, such methods might need to create a common public unlabeled dataset in the first place for the aggregation step, which could involve more complex training protocols [65].

The choice between parameter-based and output-based aggregation depends on architectural similarity of local models, communication bandwidth limit, privacy challenges, and the needs of the application.

A nuanced understanding of various aggregation strategies is crucial for designing effective and secure FL systems. To provide a high-level overview of the aggregators discussed throughout this academic work and their key characteristics, including their robustness, non-IID support, defense mechanisms, and computational considerations, we present **Table III-1**. This table serves as a quick reference, summarizing the trade-offs and design philosophies behind different aggregation approaches, which we will delve into in more detail in subsequent sections.

Table III-1:Comparison of Aggregation Strategies in Federated Learning

Aggregator	Robustness to Attack	Non-IID Support	Defense Strategy	Topology Compatibility	Computational Cost
FedAvg [11]	Low	Poor	Weighted Averaging	Flat	Low
FedNova[67]	Low	Moderate	Step-normalization	Flat	Low
Krum [68]	High	Weak	Distance-based Filtering	Flat	High
Trimmed Mean [69]	Moderate	Weak	Coordinate-wise Trimming	Flat	Moderate
Zeno [70]	High	Moderate	Suspicion-based Scoring	Flat	Moderate
FedGA [71]	Very High	Strong	Genetic Evolution Filtering	Hierarchical (edge/fog/cloud)	High

III.4. A Comprehensive Taxonomy of Model Aggregation Techniques:

The model-aggregation techniques adopted in federated learning constitute a very rich landscape, distinct for the mechanisms, pros, and cons that characterize each approach. It becomes essential to grasp the taxonomy toward selecting the correct strategy based on the peculiar constrains and aim of a given federated learning application. Basically, these techniques can be grouped according to synchronization models (synchronous vs. asynchronous), structural organizations (flat vs. hierarchical), and resilience characteristics (standard vs. robust), as shown in Figure III-1, which is elaborated comprehensively.

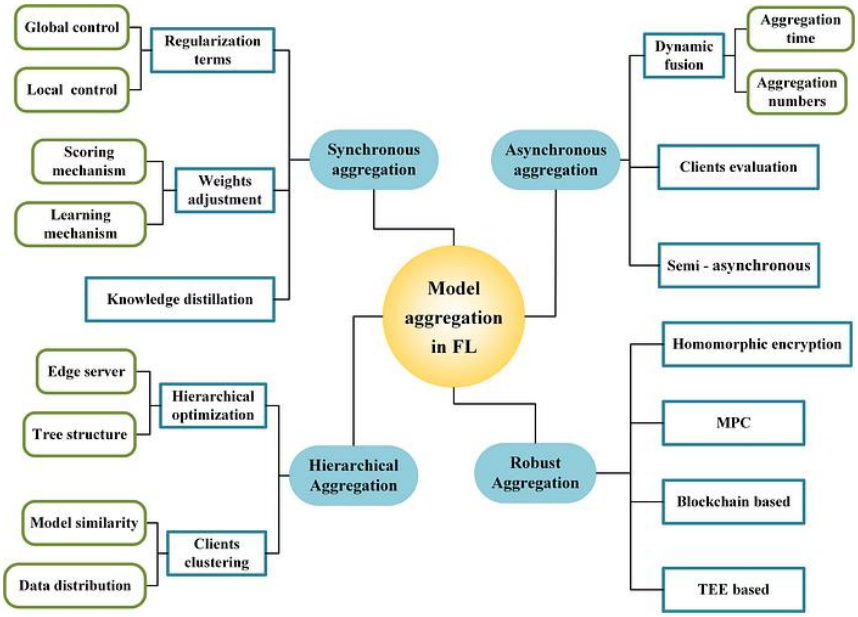


Figure III-1:A diagram of the proposed taxonomy as taken from the survey

III.4.1. Synchronous Aggregation:

Synchronous aggregation forms a basic component of the federated learning model as per round updates. This is the operation where the central server coordinates the entire updating process in which a group of clients joins in sending updates at a particular time within the window before making an update to the global model [59]. As a rule, it is also probably used in the famous FedAvg algorithm [63] because it sets up a unique synchronization barrier under which the server stops until all its chosen participants finish their local computation and send forwards their updates.

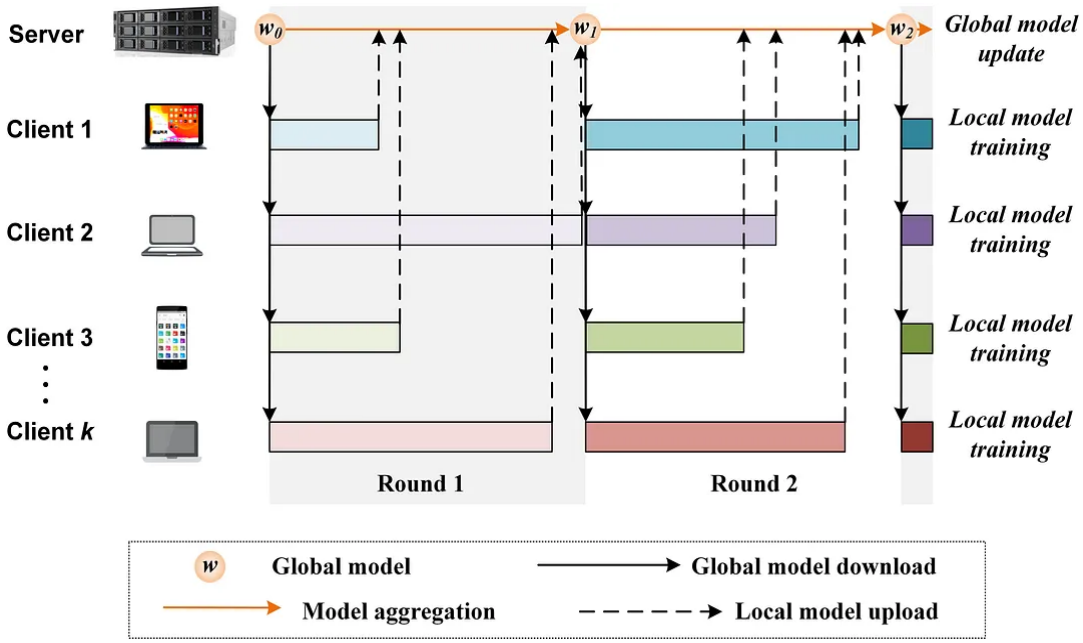


Figure III-2:Synchronous Aggregation in FL

III.4.1.1. *Mechanism and Operational Flow:*

Synchronous aggregation operates with the following cycles:

- **Client Selection:** At the beginning of the communication round t, the server picks a set of clients S_t, from among the total pool of N clients ($S_t \subseteq 1,2,\dots,N$).
- **Model Broadcast:** The server sends the current global model parameters $w_t$to all clients selected to be in S_t.
- **Local Training:** Each client $i \in S_t$independently trains the model on its local dataset D_i, resulting in local parameter updates $w_{i,t}$
- **Update Collection:** The server now waits for receiving updates from the clients in S_t, which is known as the synchronization barrier.
- **Aggregation:** With all updates now received, the server aggregates and computes parameters for the new global model for the next round w_{t+1}.

The typical aggregation formula used in FedAvg is a weighted average according to the sizes of the local datasets:

$$w_{t+1} = \sum_{i \in S_t} \frac{n_i}{\sum_{j \in S_t} n_j} w_{i,t} \quad \text{(III.5)}$$

The quantity $n_i$indicates the number of data samples with client i.

The most important restriction imposed on synchronization aggregation is that, at the start of each round of processing, the global state of the model has to be kept uniform for all

participating clients. Consequently, each selected client starts local training from exactly the same model baseline, making theoretical analysis simple and often present more predictable convergence patterns [64].

III.4.1.2. *Advantages and Disadvantages:*

Advantages:

- **Theoretical Tractability**: These methods typically offer a better approach for mathematical analysis; stronger convergence guarantees under some conditions.
- **Implementation Simplicity** Their predictable cooperative nature makes handling, debugging, and managing the entire training process much easier.
- **Consistent Model Progression**: Since these updates from clients are formed into a global now model, the global model is likely a better representation of balanced perspectives of all participating clients' data at each of the rounds.
- **Avoidance of Staleness**: All clients will use the newest version of the global model for training, eliminating problems related to old parameters.

Disadvantages:

- **The Straggler Problem**: In the straggler problem, a system's performance is held back by the slowest client. These clients, which have poor computing power or weak connectivity, may potentially postpone the training significantly.
- **Lower Fault Tolerance:** If a selected client for inclusion fails or disconnects mid-round, the server might have to wait prior to aborting the work that was already accomplished by others.
- **Inefficient Resource Use**: Faster clients need to stay idle, at the expense of a slower client, therefore the computing resources remain underutilized.
- **Scalability Challenges:** Synchronizing updates becomes complicated and inefficient with increasing numbers of clients.

III.4.1.3. *Key Variations within Synchronous Aggregation:*

FedAvg could serve as a baseline for a synchronous architecture, but it could be refined to counter difficulties such as data heterogeneity or to improve the convergence rate.

Weight Adjustment Strategies:

Such strategies escape the usual data-size weighting and embrace other factors for updating quality or relevance.

Contribution-Based Weighting: Weights are assigned depending on the inferred quality of an update (for example, by considering its impact on a validation set), historical trustworthiness of the client, or the uniqueness/diversity of the client's data.

Regularized Local Training (e.g. FedProx): FedProx [64] introduces a proximal element to the local objective function. It penalizes straying local models too much from the current global model and, by doing so, reduces the effect of clients regarding their sometimes-different updates without explicitly saying they will do it, as their data are non-ID:

$$\boldsymbol{F_i^{\text{FedProx}}(w)} = F_i(w) + \frac{\mu}{2} \parallel w - w_t \parallel^2 \quad \text{(III.6)}$$

Here, μ controls the strength of this regularization.

Adaptive Aggregation Weights (e.g., SCAFFOLD): SCAFFOLD [72] introduces control variates at both the client () and server () levels to explicitly correct for the 'client drift' caused by statistical heterogeneity. The server update incorporates these control variates:

$$w_{t+1} = w_t - \eta \left(\frac{1}{|S_t|} \sum_{i \in S_t} g_i - c + \frac{1}{|S_t|} \sum_{i \in S_t} c_i \right) \quad \text{(III.7)}$$

where g_i is the local gradient update from client i.

Incorporating Regularization Terms:

Regularization can be applied during aggregation or local training to enhance model properties.

Global Regularization (e.g., FedDyn): FedDyn [73] introduces a dynamic regularized term into the local objective function that adapts over time based on the history of model updates, aiming to align local optima with the global optimum:

$$\boldsymbol{F_i^{\text{FedDyn}}(w)} = F_i(w) + \langle h_i, w \rangle + \frac{\alpha}{2} \parallel w - w_t \parallel^2 \quad \text{(III.8)}$$

where $\boldsymbol{h_i}$ tracks the difference between local and global solutions.

Momentum-Based Aggregation (e.g., FedAvgM): Inspired by momentum in standard optimization, FedAvgM [74] applies momentum to the global model updates at the server level, potentially accelerating convergence:

$$\boldsymbol{v_{t+1}} = \beta v_t + \sum_{i \in S_t} \frac{n_i}{\sum_{j \in S_t} n_j} \left(w_{i,t} - w_t \right), \boldsymbol{w_{t+1}} = w_t + v_{t+1} \quad \text{(III.9)}$$

where β is the momentum factor and v_t is the server-side velocity vector.

Leveraging Knowledge Distillation:

Knowledge distillation is different from simple averaging parameters since it transfers knowledge using model outputs.

Server-Side Distillation (e.g., FedMD): In FedMD [66], local clients create their own models and predict results on an unlabeled public dataset. Those predictions (logits or probabilities) are sent to the server, which aggregates them (e.g., averages). Then, it trains a global "student" model on the public dataset to imitate these predicted aggregates. The objective of the server is to minimize the distillation loss:

$$\min_{w_G} \sum_{x \in D_{\text{public}}} L_{KD}\left(f(w_G; x), \frac{1}{|S_t|}\sum_{i \in S_t} f(w_i; x)\right) \quad \text{(III.10)}$$

Bidirectional Knowledge Transmission (e.g., FedDF): FedDF [75] implies distilling at the server from the aggregated client outputs on public data to import an enhanced global model which is then sent back to the clients for further localized training.

Such approaches are proved to be favorable in addressing the heterogeneity of model architectures among clients and could save up on communication costs when outputs of the model are smaller than the parameters.

III.4.2. **Asynchronous Aggregation (AFL):**

The Asynchronous Federated Learning (AFL) is quite different from the synchronized mode of its counterpart. With AFL, whenever any client sends an update, the server instantly updates the global model without enforcing any synchronization barriers [76]. The continuous update cycle allows the system to keep moving forward instead of getting stalled by slower participants.

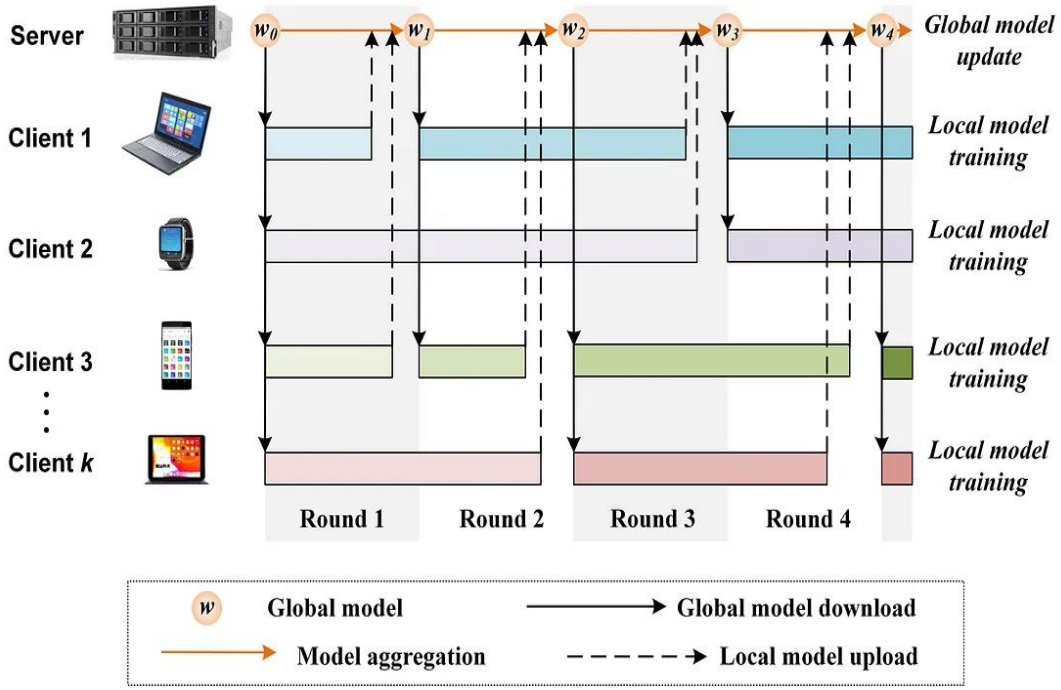


Figure III-3:Asynchronous Aggregation in FL

III.4.2.1. *Mechanism and Operational Flow:*

Under usual circumstances, the flow in AFL will look like this:

- **Initial Model Distribution:** The current global model $\boldsymbol{w_t}$ is sent by the server to any clients that are ready or request it.
- **Independent Local Training**: Each client then trains on its local data independently and at its own pace
- **Immediate Update Submission:** After finishing its own local training, client i sends the resulting model update $\boldsymbol{w_i}$ (or gradient) to the server.
- **Incremental Server Aggregation:** As soon as the server obtains the update, it is incorporated into the global model, mainly following some simple update rule like:

$$\boldsymbol{w_{t+1} = (1 - \alpha_i) w_t + \alpha_i w_i} \quad \text{(III.11)}$$

where $w_t$is the model state before update, w_{t+1} is the state after update, and α_i is a weighting factor (such as small learning rate or function of staleness).

- **Continuous Cycle:** The server keeps on accepting and integrating updates and sending down the latest global model to those clients that request it.

What really stands out about AFL is its ability to make progress without idling, even in environments that are very heterogeneous in that speeds and availability of clients vary widely [61].

III.4.2.2. *Advantages and Disadvantages:*

Advantages:

- **Straggler Elimination**: Swifter clients keep from being held back by sluggish ones, thus automatically obviating delay caused by stragglers.
- **Improved Resource Utilization**: The clients contribute straightaway as soon as they are ready for the better functioning and better use of the available computational resources and to increase the overall throughput of the system.
- **Enhanced Fault Tolerance**: The system goes on functioning with some clients failing or dropping out as it makes progress not depending on each and every client finishing tasks.
- **Improved Scalability**: AFL controls easily even with many clients, especially in cases involving irregular or unpredictable participation patterns.
- **Diminished Client Wait Times:** Clients have quick, frequent updates of the global model.

Disadvantages:

- **Update Staleness**: this is to say, clients may train with outdated versions of the global model, which can lead to inconsistencies and potentially convergence problems in case the intended updates are delayed.
- **Complex Convergence Analysis**: the theoretical analysis becomes hard with increasing staleness and absence of a synchronized global model state.
- **Possible bias**: Clients participating more often or faster may exert undue influence on the global model, pulling it towards their own local data distributions.
- **Implementation Complexity**: More advanced server-side logic will be needed to deal with concurrent updates, multiple versions of the model, and consistency.

III.4.2.3. *Strategies for Handling Update Staleness:*

To successfully mitigate the effects of staleness, the above measures are not exhaustive. Other ways include:

Staleness-Aware Weighting: reducing the contribution of stale updates in aggregation. FedAsync [77], for example, uses a function to decrease as staleness increases:

$$\boldsymbol{w_{t+1} = w_t - \eta \cdot \gamma(\tau_i) \cdot g_i} \quad \text{(III.12)}$$

where g_i is the update from client i with staleness τ_i. A simple choice is

$$\gamma(\tau) = 1/(\tau + 1). \quad \text{(III.13)}$$

Staleness-Aware Learning Rate Adaptation: Like making learning rates staleness-dependent; for example, through exponential decay [78]:

$$\boldsymbol{\eta_i = \eta_0 \cdot e^{-\beta \tau_i}} \quad \text{(III.14)}$$

III.4.2.4. *Key Variations within Asynchronous Aggregation:*

Beyond basic AFL, various approaches enhance its functionality.

Dynamic Fusion and Adaptive Weighting:

These methods, which include changing the configuration for with the combination of the client updates, introduce improvements for aggregation:

- **Adaptive Mixing Weights:** Every single update is assigned with a weight that may be manipulated according to the assignment by certain factors such as staleness, update quality, or similarity between incoming model and current global model.

- **Importance Sampling/Weighting:** Updates are weighted according to their perceived value-such as gradient magnitude, loss improvement, or contribution to data diversity [79].

Client Evaluation and Selection:

Client intelligence management can also operate asynchronously:

- **Reputation System**: Clients get scores based on their past performances (updating the quality, for example, reliability), which have an impact on whether their updates are accepted or how often their updates are chosen [80].
- **Resource-Aware Management:** The server will consider the hardware or network capabilities of the clients and perhaps tune the tasks according to their resources [81].
- **Diversity-Focused Interaction**: The system favors clients whose data can potentially enhance diversity or cover under-represented sections of the overall data distribution [82].

Semi-Asynchronous Aggregation:

Hybrid techniques that combine merits from both synchronous and asynchronous methods:

- **Periodic Synchronization** : Mainly asynchronous with synchronous points where the server waits for a minimal number of updates before aggregation [83].
- **Group-Based Synchronization** : Clients are clustered; synchronization occurs among clients in groups, whereas inter-group aggregation is asynchronous [84].
- **Adaptive Synchronization Barriers** : Sync requirements model dynamically change throughout the training, based on convergence trends or learning phase [85].

III.4.3. **Hierarchical Aggregation: Multi-Level Coordination:**

Hierarchical Federated Learning is a multi-stage architecture that is evolving from the simple star topologies attached with clients and server. Clients are usually clustered, geographically or logically grouped, with intermediate aggregators, which typically include edge servers, before producing a central server [86] for further processing. The system is expected to design improvements in scalability, lessen the load on the central server regarding communication, and efficiency.

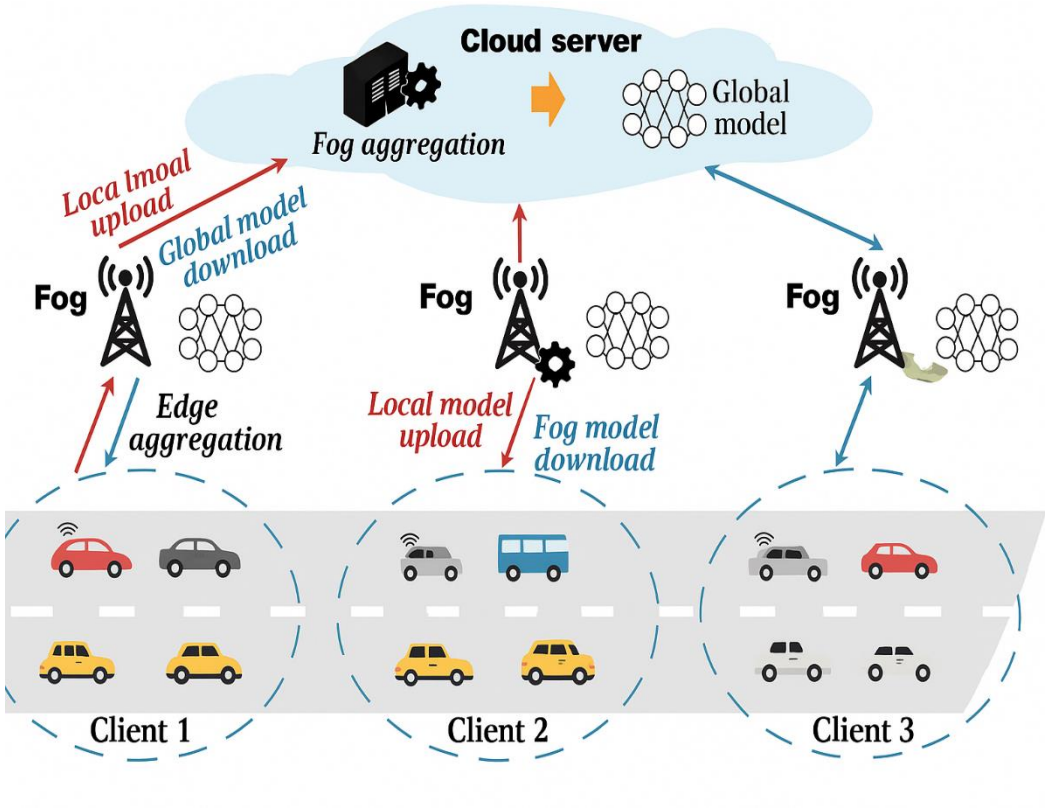


Figure III-4:Hierarchical Aggregation in FL

III.4.3.1. ***Mechanism and Operational Flow:***

Hierarchical Federated Learning (HFL) organizes clients in a tree-like structure distributing aggregation workload across different federating levels:

1. Hierarchy Definition: Multi-level architecture is created with the following:
   - Leaf nodes: Clients (for instance, edge devices),
   - Intermediate nodes: Fog aggregators,
   - Root node: A central server.
2. Intra-Cluster Aggregation: Updates are sent by clients to their own local fog aggregator.
3. Fog Aggregation: Fog nodes aggregate updates from their cluster (can be synchronous or asynchronous).
4. Inter-Cluster Communication: From fog aggregators upward-forwarded aggregated results.
5. Global Aggregation: Input storage at the fog nodes into a global model by the central server.

6. Model Dissemination: The new models are sent back down to clients in the hierarchy.

This hierarchical scheme reduces communication overheads on the central server and improves efficiencies [87].

III.4.3.2. *Advantages and Disadvantages:*

Advantages:

- **Improved Scalability**: Aggregation tasks are delegated to a large number of clients in an efficient manner.
- **Reduced Server Load**: Less computation and communication need to be performed by the central server.
- **Lower Latency:** Reduced delay due to shorter communication paths between clients and fog aggregators [88].
- **Enhanced Privacy:** Through intermediate aggregation, individual client contributions can be masked thus preserving privacy.
- **Flexibility**: Different aggregation strategies can be implemented for each level in the hierarchy.

Disadvantages:

- **Complexity Increase**: The management of multiple tiers is inherently more complicated than a flat architecture.
- **Possible Information Lost:** Aggregating locally before a global aggregation may lead to losing information regarding the fine-grained update detail.
- **Synchronization Difficulties:** Timing and consistency issues arise when coordinating within levels.
- **Security Threats:** Greater number of nodes (fog aggregators) translates to greater elevations of attack or failure.

III.4.3.3. *Variations and Considerations:*

Variations are possible in the following ways:

- **Tiers Number:** From a simple two-tier system (client-edge-server model) to more complicated three-tier architecture.
- **Aggregation Methods:** Different levels of aggregation could be synchronous, asynchronous, or robust.
- **Cluster Formation:** Clients may be grouped on the basis of location, network condition, data similarity, or administrative criteria.

- **Resource Allocation:** Right computing and networking resource allocation is essential to ensure performance and fairness at all levels.

III.4.4. **Robust Aggregation: Defending Against Adversarial Influence:**

In order to assure the robustness of the federation learning against any damages that the federated learning process may suffer in case some of the clients involved are faulty or malicious (such clients are often termed Byzantine clients), aggregation methods are being worked out [89]. These methods become paramount in order for FL to enter into the field of untrusted environments.

III.4.4.1. ***Motivations and Thread Modes:***

Robust aggregation is necessary in Federated Learning (FL) to safeguard model integrity when some client updates become faulty or malicious, aptly termed Byzantine clients. They are indispensable method for FL deployment in an untrusted environment.

In classical FL, the server assumes that updates from all its clients are trustworthy, whereas real-world situations may encounter:

- Hardware/software failure giving rise to corrupt updates.
- Malicious attacks such as:
  - Deliberately attempting to harm through submitting updates.
  - Conducting model poisoning attacks through degrading performance, injecting back doors, or skewing the model [89].

The common assumption is the number of Byzantine clients (f) is less than half of the total number of participants ($f < n/2$) so as to allow for robust detection and moderation.

III.4.4.2. ***Key Approaches to Robustness:***

There are many different categories of techniques that may be employed in robust aggregation:

Distance/Similarity-Based Filtering:

Honest updates tend to cluster together in parameter space, while Byzantine updates are outliers.

- Example: Krum, Multi-Krum
- Function: Identification and rejection of statistically distant updates.

Coordinate-Wise Robust Statistics:

Robust statistical treatments (e.g., median, trimmed mean) are applied independently on each parameter of the model.

- Example: Coordinate-Wise Median (CM), Coordinate-Wise Trimmed Mean (CTM)
- Gain: Palatable against outlier values in one dimension.

Performance/Validation-Based Filtering:

Now, when the client updates, the server checks whether the update benefits the impact when the validation set is trusted.

- Example: Zeno, FLTrust
- Method: Reject or de-weight detrimental updates in terms of validation performance.

Redundancy and Coding Techniques:

Using the theory of coding to detect or correct errors introduced by redundant updates caused by Byzantine clients.

Reputation Systems:

Long-term monitoring of clients' activities and assigning trust values to identify likely malicious beings.

- **Trade-offs in Robust Aggregation:**

Choosing a method for robust aggregation is, hence, a trade-off:

- Rigorousness of robustness guarantees
- Computational overhead
- Assumptions as regards threat models
- Influence on convergence speed during normal (i.e. nonadversarial) conditions

III.5. In-depth Analysis of Prominent Aggregators in Federated Learning:

After discussing the systematization of the various aggregation approaches, the next step delves into the details of a few prominent and widely-adopted aggregation algorithms that have influenced the federated learning domain. The focus will include, but not limited to, an inspective analysis of these algorithms, their mechanisms and mathematical formulations, major strengths, drawbacks, and the challenges that they effectively address.

III.5.1. **Foundational Aggregators:**

Early initiatives prove pivotal as foundational works for the methods of federated optimization as valid benchmarks.

III.5.1.1. ***Federated Averaging (FedAvg):***

Mechanism: A popular initiation to federated learning, FedAvg, introduced by McMahan et al [63]. operates under a synchronous update model .In each round, clients of a selected sample download the current global model and carry out multiple local training steps (epochs) on their private data using Stochastic Gradient Descent (SGD) or variations thereof, and then upload these model parameters, instead of only gradients, back to the server, which aggregates these parameters through their weighted average by the number of local data samples.

Mathematical Formulation:

1. **Server Broadcasts**: w_t is sent to selected clients S_t.
2. **Client Update**: Each client $i \in S_t$ updates w locally for E epochs, resulting in $w_{i,t}$. This involves multiple local SGD steps: $w \leftarrow w - \eta_l \nabla F_i(w)$ for each local step.
3. **Server Aggregation**: $w_{t+1} = \sum_{i \in S_t} \frac{n_i}{\sum_{j \in S_t} n_j} w_{i,t}$

Advantages:

- **Communication Efficiency**: Significantly reduces the number of communication rounds compared to sending gradients after every single local step, as clients perform multiple local updates ($E > 1$) before communicating.
- **Simplicity**: Relatively straightforward to implement and understand.

Disadvantages:

- **Client Drift**: If the local data are non-IID, there can be many local epochs (), causing the local models to drift a great deal away from the global optimum and possibly leading to convergence problems or an inferior final global model [7].
- **Sensitivity to Hyperparameters:** The effectiveness is affected by the local epochs (E) and the local learning rate (η_l).
- **Straggler Problem**: Inherits the limitations of synchronous aggregation.

III.5.1.2. ***Federated Stochastic Gradient Descent (FedSGD) :***

Mechanism: FedSGD is simply a variant of FedAvg in which the clients do exactly one local update (or work with a single mini-batch) before sending their respective updates-usually gradients or parameter deltas-to the server. This corresponds more closely with conventional distributed SGD.

Mathematical Formulation:

1. **Server Broadcasts**: w_t is sent to selected clients S_t.
2. **Client Computation**: Each client $i \in S_t$ computes a gradient $g_{i,t} = \nabla F_i(w_t)$ based on a mini-batch of its local data.
3. **Server Aggregation**: $g_t = \sum_{i \in S_t} \frac{n_i}{\sum_{j \in S_t} n_j} g_{i,t}$ $w_{t+1} = w_t - \eta_g g_t$ (where η_g is the global learning rate).

Advantages:

- **Simpler Analysis**: Closer to standard distributed SGD, making theoretical analysis potentially easier.
- **Reduced Client Drift**: Minimizes the divergence of local models compared to FedAvg with $E > 1$.

Disadvantages:

- **High Communication Cost**: Requires significantly more communication rounds than FedAvg, as updates are sent after each local step.
- **Straggler Problem**: Still subject to synchronous bottlenecks.

III.5.2. Aggregators Tackling Statistical Heterogeneity:

Non-IID data is a defining challenge in FL. These methods aim to improve convergence and accuracy in such settings.

III.5.2.1. *FedProx: Addressing Heterogeneity via Regularization:*

Mechanism proposed by Li et al. [64] described that FedProx customizes the local client objective function with an extra proximal term. This term penalizes for large divergence of local model parameters in comparison to current global model parameters, thereby regularizing local training and minimizing client drift.

Mathematical Formulation:

1. **Local Objective Modification**: Client i minimizes: $h_i(w; w_t) = F_i(w) + \frac{\mu}{2} \| w - w_t \|^2$ where $F_i(w)$ is the standard local loss, w_t is the global model from the start of the round, and $\mu \geq 0$ is a hyperparameter controlling the strength of the proximal term.
2. **Aggregation**: Standard FedAvg aggregation is used on the resulting local models.

Advantages:

- **Improved Convergence on Non-IID Data**: Theoretically and empirically shown to improve convergence stability compared to FedAvg in heterogeneous settings.
- **Handles System Heterogeneity**: The formulation allows for variable amounts of local work across clients (inexact solutions to the local objective), making it more robust to system heterogeneity.
- **Simple Modification**: Requires only a minor change to the client-side local objective function.

Disadvantages:

- **Hyperparameter Tuning**: Requires tuning the proximal parameter μ, which can be data-dependent.
- **Potential Slowdown**: The proximal term might slightly slow down convergence in IID settings compared to FedAvg if μ is too large.

III.5.2.2. *Genetic Algorithms (FedGA):*

Mechanism: Federated Learning using Genetic Algorithms (FedGA) employs evolutionary principles to enhance model aggregation, moving beyond simple averaging or regularization. As detailed by Guendouzi et al. [71] and Zheng et al. [90], FedGA treats local models or updates as individuals in a population undergoing evolutionary steps across rounds:

- **Representation:** Local models ($w_{i,t}$) or updates are encoded as chromosomes (e.g., parameter vectors).
- **Fitness Evaluation:** A fitness function (f) assesses each model's quality based on metrics like accuracy, convergence speed, robustness, or generalization on non-IID data.
- **Selection:** Fitter individuals are selected for reproduction using methods like roulette wheel or tournament selection.
- **Crossover:** Selected models ("parents") exchange components (e.g., layers, weights) to create hybrid offspring models.
- **Mutation:** Small random changes are introduced to offspring parameters to maintain diversity and explore the solution space.
- **New Generation/Aggregation:** The best individual(s) from parents and offspring form the new global model (w_{t+1}) or the population for the next round.

The core process of the Federated Genetic Algorithm, encompassing both server-side aggregation and client-side updates, is formally described in Algorithm :

Algorithm 2 Federated Genetic Algorithm (FedGA)

The **K** clients are indexed by **k**; **B** is the local mini-batch size, $\mathcal{D}_k$is the dataset available to client **k**, $\mathcal{D}_t$ is the dataset used for the test which is available on the aggregator, $W_{\mathbf{B}}$ is the vector of base layers, $W_{\mathbf{P}}$ is the vector of personalized layers, **E** is the number of local epochs, and η is the learning rate.

Procedure FedGA ▷ Run on the server.
Initialize $W^0_{\mathbf{B}}$;
for `each round` $\mathtt{t} = 1, 2, 3, \ldots$ **do**
  for `each client` $\mathtt{k} \in \mathtt{K}$ **do**
    $W^{t+1}_{\mathbf{B},\mathbf{k}} \leftarrow ClientUpdate(\mathrm{k}, \mathrm{W}^t_{;bfB,k})$; ▷ In Parallel.
  end for
  $W^{t+1}_{\mathbf{B}} = GA(\mathcal{D}_t, W^t_{\mathbf{B}})$; ▷ Only base layers are aggregated
end for
End procedure FedGA
Procedure ClientUpdate(k, w^t_B) ▷ Run on client **k**.
$\beta \leftarrow$ (Split $\mathcal{D}_k$into mini-batches of size B;)
for each local epoch **i** from 1 to **E** **do**
  for batch $\mathbf{b} \in \beta$ **do**
    $w_{\mathbf{B}}, w_{\mathbf{P}} \leftarrow w - \eta \Delta \mathcal{L}(w_{\mathbf{B}}, w_{\mathbf{P}}, b)$; ▷ base layers are updated and trained and personnalized layers are trained
  end for
end for
`return t to the Server`
End procedure ClientUpdate$(k, w_{\mathbf{B}})$

Figure III-5:Federated Genetic Algorithm (FedGA)[71]

Advantages:

- **Advanced Performance Prospects**: Intelligent exploration of the model space may meaningfully accelerate convergence and capture higher accuracy for problems with latitudes of complexities or heterogeneous data [90], [91].
- **Multifunction Optimization:** The adaptability of these functions allows for fitness functions considering more objectives (robustness, model size, fairness,) than just standard loss minimization.
- **Adapted for Heterogeneity Management**: GAs probably have the possibility of accounting for all kinds of statistical and system heterogeneities just by solving those very adapted model selection problems.
- **Scalability Stability**: Preliminary studies suggest that the GA-based methods might actually have a better chance of maintaining stability while increasing numbers of clients [90].

Disadvantages:

- **Computational Complexity:** GA operations (fitness evaluation, crossover, mutation) add server-side overhead compared to simpler methods.
- **Hyperparameter Tuning:** Requires careful tuning of GA parameters (population size, rates, fitness function, selection method).
- **Communication Cost:** Fitness evaluation might necessitate additional client-server communication.
- **Novelty & Standardization:** As an emerging approach, FedGA lacks standardization, requiring further validation across diverse scenarios.

III.5.2.3. *FedNova: Normalizing Local Updates:*

Mechanism: Unlike tailoring local training, which is the common method of combating heterogeneity, Wang et al. [92] have proposed to focus primarily on the aggregation mechanism. It is observed that due to the reason of system heterogeneity or clients' own choice, when clients run a different number of local steps, direct averaging of their model parameters in FedAvg leads to an incorrect weightage. FedNova equalizes local updates in relation to the amount of local work done prior to averaging.

Mathematical Formulation:

1. **Client Update**: Client i performs τ_i local steps (e.g., gradient updates) starting from w_t, resulting in $w_{i,t}$. The local update is $a_{i,t} = w_{i,t} - w_t$.
2. **Server Aggregation**: The server computes a normalized global update: $w_{t+1} = w_t - \eta_g \sum_{i \in S_t} \frac{n_i}{n} \frac{a_{i,t}}{\tau_i}$ Alternatively, it can be viewed as adjusting the aggregation weights: $w_{t+1} = w_t + \sum_{i \in S_t} p_i \frac{a_{i,t}}{\tau_i} \left(\sum_{k \in S_t} p_k \tau_k \right)$ where $p_i = n_i / n$.

Advantages:

- **Corrects for Heterogeneous Local Work**: Properly handles scenarios where clients perform different numbers of local updates, which FedAvg fails to do.
- **Improved Convergence**: Empirically shown to converge faster and to better solutions than FedAvg, especially when local steps vary across clients.
- **No Change to Local Training**: Does not require modifying the client-side training process.

Disadvantages:

- **Requires Tracking Local Steps**: Clients need to report the number of local steps (τ_i) performed to the server.
- **Focus on System Heterogeneity**: Primarily addresses inconsistencies arising from varying amounts of local computation, less directly targeting statistical heterogeneity compared to FedProx.

III.5.3. **Robust Aggregators: Defending Against Adversarial Clients:**

These methods prioritize resilience against faulty or malicious clients that might attempt to disrupt the training process.

III.5.3.1. ***Krum / Multi-Krum: Distance-Based Selection:***

Mechanism: Krum [40] holds that honest client updates are clustered together, while malicious updates are distributed as outliers. It selects that single client update that minimises the sum of the squared Euclidean distances to its nearest neighbours, where is the number of participating clients and represents the maximum number of Byzantine attackers assumed. Multi-Krum generalises this by selecting the lowest Krum scores and averaging them across the clients with the highest scores.

Mathematical Formulation (Krum):

1. Compute pairwise squared distances: $d_{ij} = \| w_{i,t} - w_{j,t} \|^2$ for all $i, j \in S_t$.
2. For each client i, find the set N_i of its $n - f - 2$ nearest neighbors (excluding itself).
3. Compute Krum score for each client: $s_i = \sum_{j \in N_i} d_{ij}$.
4. Select the client with the minimum score: $i^* = \text{argmin}_{i \in S_t} s_i$.
5. Use the selected update: $w_{t+1} = w_{i^*,t}$ (or $w_{t+1} = w_t - \eta_g (w_t - w_{i^*,t})$).

Advantages:

- **Provable Robustness**: Provides theoretical guarantees against up to f Byzantine attackers (under certain assumptions).
- **No Server Data Needed**: Does not require a trusted validation set on the server.

Disadvantages:

- **High Computational Cost**: Requires computing $O(n^2)$ pairwise distances, which is expensive for large n. The complexity is $O(n^2 d)$, where d is the model dimension.
- **Information Loss**: Krum discards updates from all but one (or m) clients, potentially losing valuable information from honest clients, especially if data is non-IID.
- **Assumption on Attackers**: Assumes $f < n/2$.

III.5.3.2. ***Coordinate-Wise Median / Trimmed Mean: Statistical Robustness:***

Mechanism: These methods apply robust statistical estimators independently to each coordinate (parameter) of the model updates [93].

Coordinate-Wise Median (CM): For each parameter j, the server collects the j-th coordinate value from all client updates ($w_{i,t}^{(j)}$ for $i \in S_t$) and computes their median. The aggregated global update's j-th coordinate is this median value.

Coordinate-Wise Trimmed Mean (CTM): For each parameter j, the server sorts the values $w_{i,t}^{(j)}$, removes a β fraction of the lowest and highest values, and computes the mean of the remaining $(1-2\beta)n$ values.

Mathematical Formulation (CM):

$$w_{t+1}^{(j)} = \text{median}\{w_{i,t}^{(j)}\}_{i \in S_t} \quad \text{(III.15)}$$

Mathematical Formulation (CTM): Let S_j be the sorted list of values $\{w_{i,t}^{(j)}\}_{i \in S_t}$. Let L_j and H_j be the indices corresponding to the $\lfloor \beta n \rfloor$ lowest and highest values. $w_{t+1}^{(j)} = \frac{1}{n-|L_j|-|H_j|} \sum_{k \notin L_j \cup H_j} S_j\,[k]$

Advantages:

- **Computational Efficiency**: Generally more computationally efficient than distance-based methods like Krum, especially CTM (sorting takes $O(n\log n)$ per coordinate, total $O(dn\log n)$).
- **Robustness**: Provides robustness against outliers affecting specific coordinates.
- **No Server Data Needed**: Operates solely on client updates.

Disadvantages:

- **Potential Information Distortion**: Applying median or trimming coordinate-wise might distort the relationship between parameters compared to the original updates.
- **Weaker Guarantees than Krum**: May have weaker theoretical guarantees against sophisticated, coordinated attacks compared to Krum under certain conditions.
- **Requires Sufficient Clients**: Statistical estimators require a reasonable number of clients (n) to be effective.

III.5.3.3. *FLTrust: Trust Bootstrapping via Server Validation:*

FLTrust uses a small, trusted dataset referred to as root dataset that only the server possesses. This dataset is used to compute a 'trusted' gradient or update direction () with respect to the current global model. It then calculates the cosine similarity between each client's update () and the trusted direction. These similarity scores are normalized (using ReLU) and treated as trust weights () to compute a weighted average of client updates.

Mathematical Formulation:

1. Server computes trusted update: $g_{\text{root}} = \nabla F_{\text{root}}(w_t)$.
2. Clients compute local updates: $g_i = w_{i,t} - w_t$.
3. Server computes trust scores: $TS_i = \text{ReLU}\big(\cos(g_i, g_{\text{root}})\big)$.
4. Server normalizes scores: $\overline{TS_i} = TS_i / \sum_{k \in S_t} T\,S_k$.

5. Server aggregates: $w_{t+1} = w_t + \eta_g \sum_{i \in S_t} \overline{TS}_i \, g_i$.

Advantages:

- **Effective Against Model Poisoning**: Empirically shown to be robust against sophisticated model poisoning attacks that might evade distance-based methods.
- **Leverages Server Knowledge**: Uses the server's small trusted dataset effectively.
- **Moderate Computational Cost**: Primarily involves cosine similarity calculations.

Disadvantages:

- **Requires Trusted Server Dataset**: Relies on the availability and quality of a small, representative dataset on the server, which might not always be feasible.
- **Potential Bias**: The aggregation can be biased towards the server's root dataset.

III.5.3.4. *Zeno: Performance-Based Filtering:*

That is Zeno [49]: Requires a small validation dataset at the server, where it estimates the impact on the validation loss if each client's update was included. In particular, it assigns to each client a score that represents the estimated descent on the validation loss due to application of that client's update, minus a penalty term related to the update's magnitude. Rejected or down-weighted will be those updates whose scores are under a certain threshold (possibly negative, indicating degradation).

Mathematical Formulation:

1. Server computes gradient on validation set: $g_{\text{val}} = \nabla F_{\text{val}}(w_t)$.
2. Clients compute local updates: $g_i = w_t - w_{i,t}$.
3. Server computes score for client i: $s_i = -\eta_g \langle g_{\text{val}}, g_i \rangle - \frac{\rho}{2} \parallel \eta_g g_i \parallel^2$ (This estimates the change in validation loss: $\approx F_{\text{val}}(w_t - \eta_g g_i) - F_{\text{val}}(w_t)$).
4. Server selects clients with positive scores: $S'_t = \{i \in S_t \mid s_i > 0\}$.
5. Server aggregates using selected updates (e.g., FedAvg on S'_t).

Advantages:

- **Direct Performance Optimization**: Directly links robustness to the impact on validation performance.
- **Robustness Guarantees**: Provides theoretical robustness guarantees.

Disadvantages:

- **Requires Trusted Server Dataset**: Similar to FLTrust, it depends on a server-side validation set.
- **Computational Cost**: Requires computing gradients on the validation set and inner products for each client update.
- **Hyperparameter Tuning**: Involves tuning the penalty parameter ρ.

III.6. Conclusion:

In this chapter, we provided a comprehensive overview of aggregation techniques in Federated Learning, with a focus on addressing key challenges such as data heterogeneity (Non-IID distributions) and the need for robustness against system failures and adversarial behavior. We examined both traditional methods like FedAvg and more advanced approaches including FedProx, FedNova, FedGA, as well as robust aggregation algorithms like Krum and Trimmed Mean.

This chapter has laid the theoretical foundation for understanding the design and purpose of these aggregators. In the following chapter, we move toward the experimental evaluation of these techniques, presenting the results obtained in terms of accuracy, communication efficiency, and resilience across different federated learning scenarios.

Chapter 4: Implementation and Experiments

IV.1. Introduction:

In the previous chapters, we explored the foundations of federated learning, its challenges in terms of privacy protection, and the various attacks that can compromise the integrity and confidentiality of data. We also analyzed different aggregation architectures and their vulnerabilities.

In this final chapter, we first present the overall architecture and the tools used for our federated learning implementation. Then, we describe the attack tests conducted on different aggregation protocols, with a focus on the central aggregators. The results highlight the real security risks associated with these models, confirming the need for more robust and secure solutions.

IV.2. Work Environment:

IV.2.1. Hardware Environment:

For the development and testing of our approach, two computers were used:

- **Computer 1 (HP):**
  - Processor: Intel(R) Core (TM) i5-8350U CPU @ 1.90 GHz
  - Memory: 8 GB
  - Operating System: Windows 10 64-bit
- **Computer 2: *(*dell*):***
  - Processor: Intel(R) Core (TM) i5-8350U CPU @ 1.90 GHz
  - Memory: 8 GB
  - Operating System: Windows 10 64-bit

IV.2.2. Runtime Environment:

IV.2.2.1. *Google Colab:*

Google Colab (or Google Colaboratory) is a cloud-based computing platform developed by Google that enables the execution of Jupyter notebooks in a hosted environment. It provides free access to computational resources, including Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), making it a valuable tool for the development and training of models in artificial intelligence, machine learning, and data analysis. Thanks to its integration with Google Drive and compatibility with programming languages such as Python and R, Google Colab facilitates real-time collaboration, reproducibility of experiments, and dissemination of scientific results

IV.2.2.2. *Amazon SageMaker:*

Amazon SageMaker is a fully managed machine learning platform developed by Amazon Web Services (AWS), enabling researchers and developers to build, train, and deploy machine learning models at scale in a cloud environment. It provides an integrated suite of tools that streamline every stage of the machine learning lifecycle, from data preprocessing to production inference. With its elastic computing capabilities, seamless integration with other AWS services, and optimized Jupyter notebook environments, SageMaker serves as a robust solution for advanced research and industrial applications in artificial intelligence.

IV.2.2.3. *Programming languages and libraries:*

Python:

Python is an interpreted, general-purpose, object-oriented programming language widely used in scientific computing, data analysis, artificial intelligence, and software development. It's simple and readable syntax makes it a particularly attractive tool for academic research, especially given its rich ecosystem of specialized libraries such as NumPy, Pandas, Scikit-learn, TensorFlow, and Matplotlib. Within the framework of a master's research project, Python stands out as a preferred choice for algorithmic implementation, statistical modeling, and the reproducibility of experimental results.

PyTorch:

Pytorch is an open source machine learning framework based on the Torch 2 library for Python programming language, developed by Facebook's AI Research lab (FAIR) in 2016. It can be used on cloud platforms and provides the elegantly designed modules and classes torch.nn, torch.optim, Dataset, and DataLoader to help you create and train neural networks.

Numpy:

NumPy (short for Numerical Python) is a core Python library dedicated to numerical computing and the efficient manipulation of large multidimensional arrays of homogeneous data. It serves as the foundation for mathematical and statistical operations in data analysis and machine learning environments. Within the framework of a master's research project, **NumPy** plays a fundamental role in implementing high-performance numerical algorithms while supporting methodological rigor and reproducibility in experimental research.

Matplotlib:

Matplotlib is a 2D and 3D data visualization library for the Python programming language, widely used in scientific and engineering fields to graphically represent numerical results, time series, statistical distributions, or mathematical models. It enables the production

of high-quality figures that can be easily integrated into academic papers or research reports. Within the framework of a master's research project, **Matplotlib** stands out as an essential tool for exploratory data analysis, model validation, and visual communication of experimental findings.

Pygad:

PyGAD is an open-source Python library designed for the implementation of **genetic algorithms (GA)**, a class of metaheuristics inspired by the biological process of natural evolution. It enables the resolution of complex optimization problems, particularly in cases involving non-linear, discontinuous, or multi-modal functions, where classical deterministic methods reach their limits. Thanks to its user-friendly interface and modular architecture, **PyGAD** is suitable for researchers in machine learning, engineering, and applied sciences, facilitating the integration of evolutionary algorithms into academic or industrial projects.

IV.2.2.4. *Development tools*

Vscode:

Visual Studio Code, commonly referred to as **VS Code**, is a free and open-source code editor developed by Microsoft. It is a versatile and extensible tool widely adopted in scientific, technical, and computer science fields for software development, data analysis, and artificial intelligence programming. With advanced features such as auto-completion, integrated debugging, version control with Git, and broad compatibility across multiple programming languages (including Python), **VS Code** provides a powerful and customizable working environment for implementing and testing algorithms in both academic and industrial settings.

IV.3. The architecture used in this project:

This section presents the Edge-Fog-Cloud hierarchical architecture used as the experimental framework in this project. Composed of three layers — Edge (local devices), Fog (intermediate aggregation), and Cloud (global aggregation) — this architecture enables collaborative learning while preserving data privacy. It was selected not only for its practical relevance in federated learning systems, but also to serve as a **realistic attack surface** for evaluating various **privacy attacks**. This multi-tiered structure provides a solid foundation for analyzing the system's vulnerability to threats.

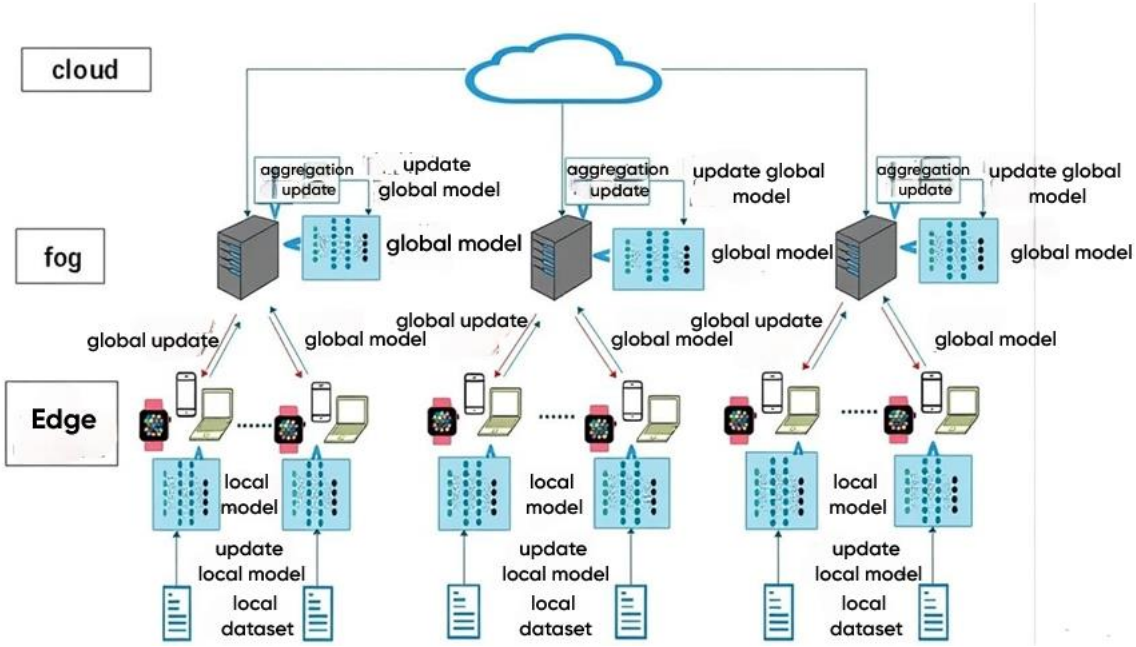


Figure IV-1:Schematic Representation of the Hierarchical Architecture in Federated Learning with Edge, Fog, and Cloud

Within the context of distributed federated learning, the hierarchical Edge-Fog-Cloud architecture offers a structured and efficient solution to address challenges related to latency, bandwidth consumption, and data privacy. This three-tiered framework consists of interconnected layers: end-user devices (Edge), intermediate aggregation nodes (Fog), and a central coordination server (Cloud).

At the **Edge** level, individual clients perform local model training on their own private datasets, in accordance with the core principle of federated learning—minimizing data exposure by transmitting only model updates rather than raw data. Upon completing local training over multiple epochs, clients send their updated model weights to their assigned Fog node.

The **Fog** layer serves as an intermediary aggregator. It collects model updates from connected Edge clients, performs local aggregation, and forwards the resulting partially updated model to the Cloud. This step reduces communication overhead toward the central server while mitigating local variations and improving system robustness.

At the top layer, the **Cloud** coordinates the overall learning process. It selects participating Fog nodes, broadcasts the global model, collects locally aggregated models, and applies a chosen aggregation strategy (e.g., FedAvg, FedNova) to update the global model. The updated model is then evaluated on a centralized test dataset to monitor performance and detect potential anomalies or malicious behaviors, such as targeted scaling attacks.

The system operates through an iterative cycle that includes model distribution, local training, local aggregation, global aggregation, and final evaluation. This loop repeats for a predefined number of rounds until the global model converges toward an optimal solution.

This hierarchical architecture provides several key advantages: reduced bandwidth usage through intermediate aggregation, enhanced privacy preservation by keeping raw data at the Edge, increased fault tolerance due to its modular structure, and improved resilience against adversarial attacks through robust aggregation techniques.

IV.4. Dataset:

MNIST (for *Modified National Institute of Standards and Technology database*) is a widely used dataset in machine learning and artificial intelligence. It contains 70,000 grayscale images of handwritten digits (from 0 to 9), each with dimensions of 28 × 28 pixels, labeled according to their respective class. This dataset is often considered a standard benchmark for evaluating low-level image classification algorithms.

Fashion-MNIST, developed by Zalando Research, is a modern alternative to MNIST. It shares the same dimensional characteristics as MNIST (60,000 training images and 10,000 test images) but replaces digits with grayscale images of clothing items and accessories belonging to 10 distinct classes (e.g., T-shirt, trousers, shoes, etc.). Fashion-MNIST has become a relevant benchmark for assessing the performance of supervised learning models in scenarios where visual complexity exceeds that of the MNIST dataset.

IV.5. Neural Network Models:

The neural network model used for the experiments is a **convolutional neural network (CNN)** designed to process grayscale images of size `28x28`, such as those from the MNIST or FashionMNIST datasets. It consists of three convolutional layers followed by max-pooling layers: the first layer applies 32 filters of size `(3,3)` with a ReLU activation function, the second uses 64 identical filters, and the third increases to 128 filters to capture more abstract features. Spatial dimensions are progressively reduced through max-pooling operations `(2,2)`. After the convolutional layers, the data is flattened and passed through a dropout layer (rate = 0.5) to prevent overfitting before reaching a fully connected layer with 10 units and a softmax activation, corresponding to the output classes. The model is trained using the Adam optimizer (`learning_rate = 0.001`), categorical cross-entropy loss, and accuracy as the metric. This architectural choice offers a good balance between performance, simplicity, and robustness, making it an ideal candidate for evaluating the impacts of Scaling attacks in a hierarchical Edge-Fog-Cloud federated system.

```python
def create_simple_cnn():
    model = keras.Sequential(
        [
            keras.Input(shape=(28, 28, 1)),
            keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding="same"),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Flatten(),
            keras.layers.Dropout(0.5),
            keras.layers.Dense(10, activation="softmax"),
        ]
    )
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])
    return model
```

Figure IV-2:simple CNN model.

In addition to the CNN, a Multi-Layer Perceptron (MLP) model was also implemented for comparison purposes and to evaluate the robustness of aggregation strategies across different model architectures. This MLP is specifically designed to process flattened input data, making it suitable for datasets like MNIST or FashionMNIST.

The MLP model's architecture, as implemented in our experiments, consists of an input layer, two hidden layers, and an output layer. The input layer receives data with `input_dim` features (e.g., 28×28=784 for MNIST images). This is followed by the first hidden layer, which contains `hidden_dim` (e.g., 128) neurons, applying a ReLU activation function. The output of this layer then feeds into a second hidden layer, also with `hidden_dim` (e.g., 128) neurons, similarly using a ReLU activation. Finally, the model outputs to a fully connected layer with `num_classes` (e.g., 10) units, corresponding to the total number of output classes, typically with a softmax activation implicitly handled by the loss function for classification tasks.

The overall structure of this Multi-Layer Perceptron, illustrating the flow from input to output through its hidden layers, can be visualized as follows:

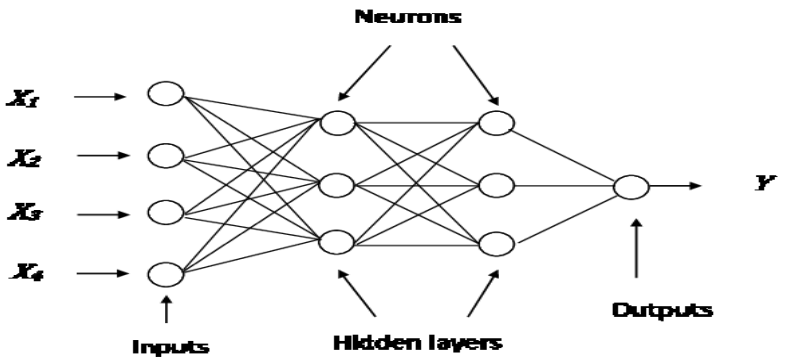


Figure IV-3:Topology of the Multi-Layer Perceptron (MLP) model used in experiments

IV.6. Performance Evaluation

A variety of performance metrics were used to evaluate the proposed model, including accuracy, precision, recall, and the F1-score**.** This systematic comparative evaluation with other relevant approaches provides a deeper understanding of our model's effectiveness. The metrics are described below:

IV.6.1. **Precision:**

Precision measures the proportion of correctly predicted positive instances among all instances predicted as positive. It reflects the model's ability to avoid false positives.

$$Precision = \frac{tp}{tp+fp} \quad \text{(IV.1)}$$

IV.6.2. **Recall (Sensitivity or True Positive Rate)**

Recall quantifies the proportion of actual positive instances that were correctly identified by the model. It indicates how well the model captures positive cases.

$$Recall = \frac{tp}{tp+fn} \quad \text{(IV.2)}$$

IV.6.3. **F1-Score:**

The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance, especially in cases of class imbalance.

$$F1 - score = 2 \cdot \frac{Precision+Recall}{Precision \cdot Recall} \quad \text{(IV.3)}$$

IV.6.4. **Accuracy:**

Accuracy represents the ratio of correctly classified instances (both positive and negative) to the total number of instances. It gives an overall indication of model correctness.

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn} \quad \text{(IV.4)}$$

IV.7. Attack Methodology:

This section presents the detailed results of our experiments, focusing on the impact of different types of attacks on the main task accuracy of federated learning models and on the privacy of training data. We analyze the performance of various scenarios and aggregation strategies under both benign and adversarial conditions.

IV.7.1. **Label flip attack:**

IV.7.1.1. ***Overview of the Attack:***

The Label Flipping Attack is a form of data poisoning attack in federated learning, where malicious clients intentionally modify the labels of their local training data to mislead the global model during aggregation. This type of attack can significantly degrade the performance of the

global model by introducing label inconsistencies without requiring direct manipulation of model updates. To emulate this threat, we implemented a Python class named LabelFlipAttack, which inherits from a base class BaseAttack. The core functionality involves flipping a fraction of labels either to a specific target label or randomly to any other class, depending on the configuration.

IV.7.1.2. *Key Attack Parameters:*

Table IV-1:Summary of Experimental Parameters for label flip Attack

parameter	value	Description
Dataset	MNIST	A standard dataset consisting of grayscale images of handwritten digits (28x28 pixels), with 10 classes (digits 0 to 9).
Model Architecture	MLP	A fully connected feedforward neural network used by Edge clients and the global server
Data Distribution	Non-IID via Dirichlet ($\alpha = 0.5$)	Data is distributed among clients using a Dirichlet distribution with concentration parameter $\alpha = 0.5$, resulting in a realistic label skew across clients.
Number of Clients	10	Total number of participating Edge devices in the federated learning process.
Number of Fog Nodes	2	Intermediate aggregation layer between Edge clients and the Cloud server
Global Communication Rounds	20	Number of communication cycles between the Cloud and Fog layers during training
Local Epochs	1	Each Edge client trains its local model for one full pass over its assigned dataset before submitting an update to the Fog node.
Batch Size	32	Mini-batch size used during local training on Edge clients.
Learning Rate (local)	0.01	Step size used by the Adam optimizer during local training

Attack Type	Label Flipping Attack	An adversarial attack where a subset of clients flips their local labels
Target Label (for attack)	Class 5	in this scenario, samples originally labeled as class 5 are maliciously relabeled as class 7
Fog Aggregator	FedAvg	The Fog layer uses Federated Averaging to combine local model updates before sending them to the Cloud for global aggregation.
Cloud Aggregators Evaluated	FedAvg, Krum, TrimmedMean (β=0.4), Zeno (ρ=0.1, k=0.7), FedGA	five aggregation strategies tested

IV.7.1.3. ***Experimental Results:***

Figure IV-4 illustrates the evolution of accuracy for different cloud aggregators under a Label Flip Attack with 30% malicious clients. The results show that:

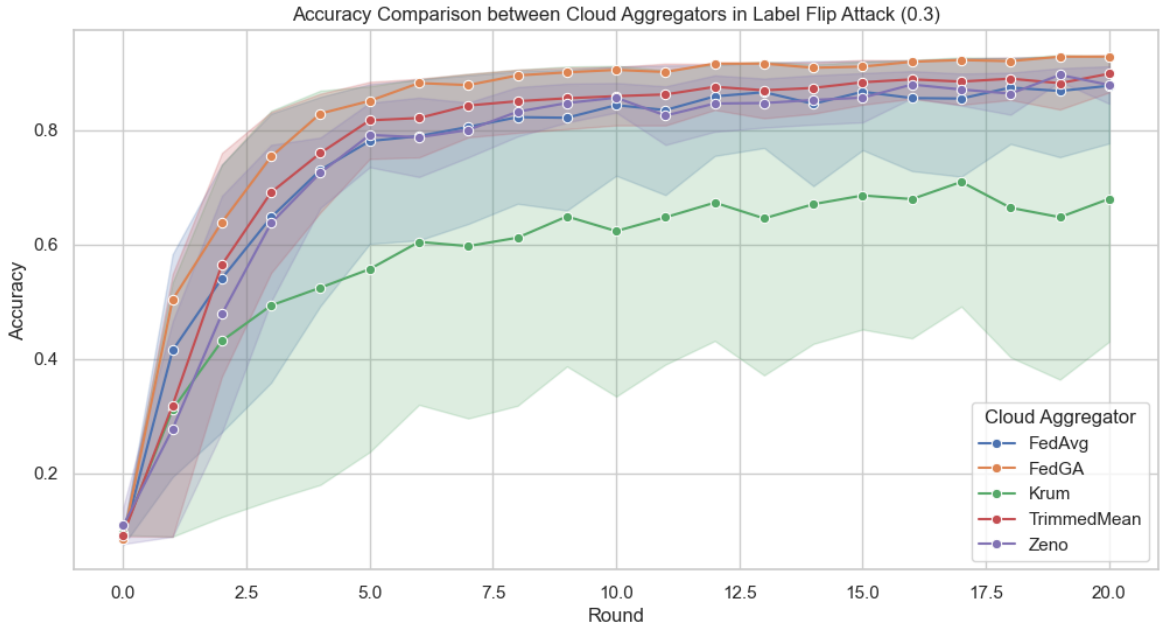


Figure IV-4:Comparison of Accuracy Among Cloud Aggregators Under Label Flip Attack (30% Malicious Clients)

This figure illustrates the evolution of the accuracy of the different cloud aggregators in the face of a "Label Flip" attack with 30% malicious clients. We observe that FedAvg and TrimmedMean maintain high accuracy, reaching levels similar to those of the benign scenario. Zeno and FedGA also demonstrate good resilience, with accuracy converging towards high values. Krum, on the other hand, exhibits significantly lower accuracy, indicating its vulnerability to this level of attack.

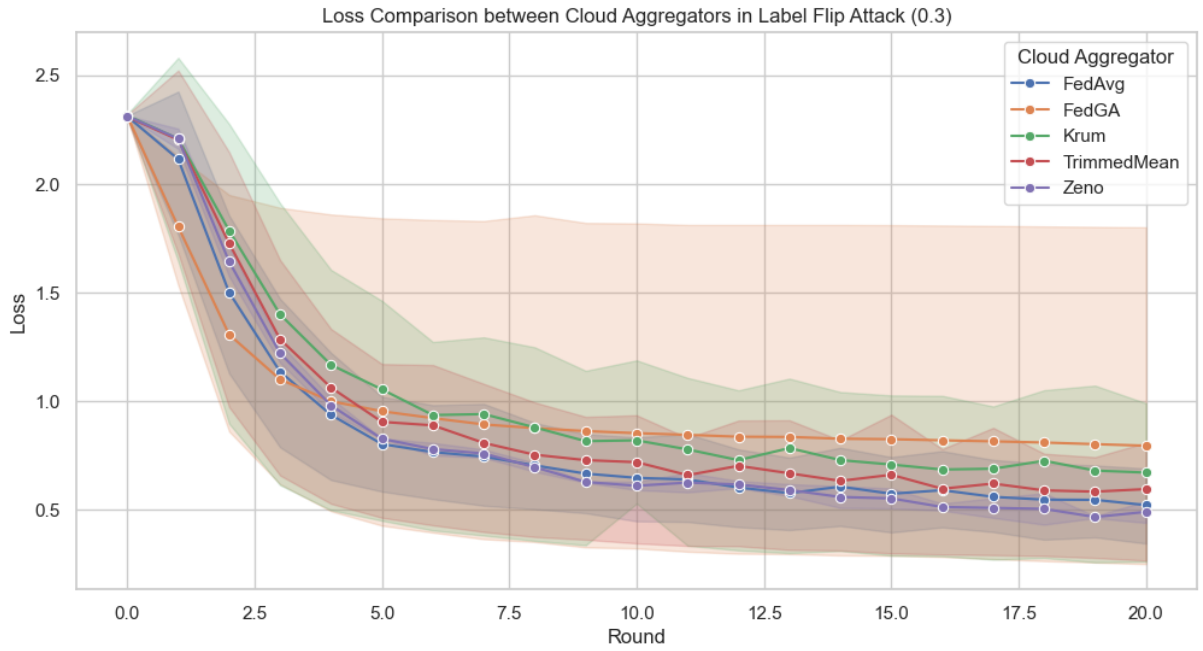


Figure IV-5:Comparison of Loss Among Cloud Aggregators Under Label Flip Attack (30% Malicious Clients)

Figure IV-5 illustrates the evolution of loss for different aggregation methods under the same attack. FedAvg **,** TrimmedMean **,** Zeno , and FedGA manage to significantly reduce the loss over rounds, indicating good model convergence despite the attack. In contrast, Krum shows higher loss values and slower convergence, confirming its lower robustness.

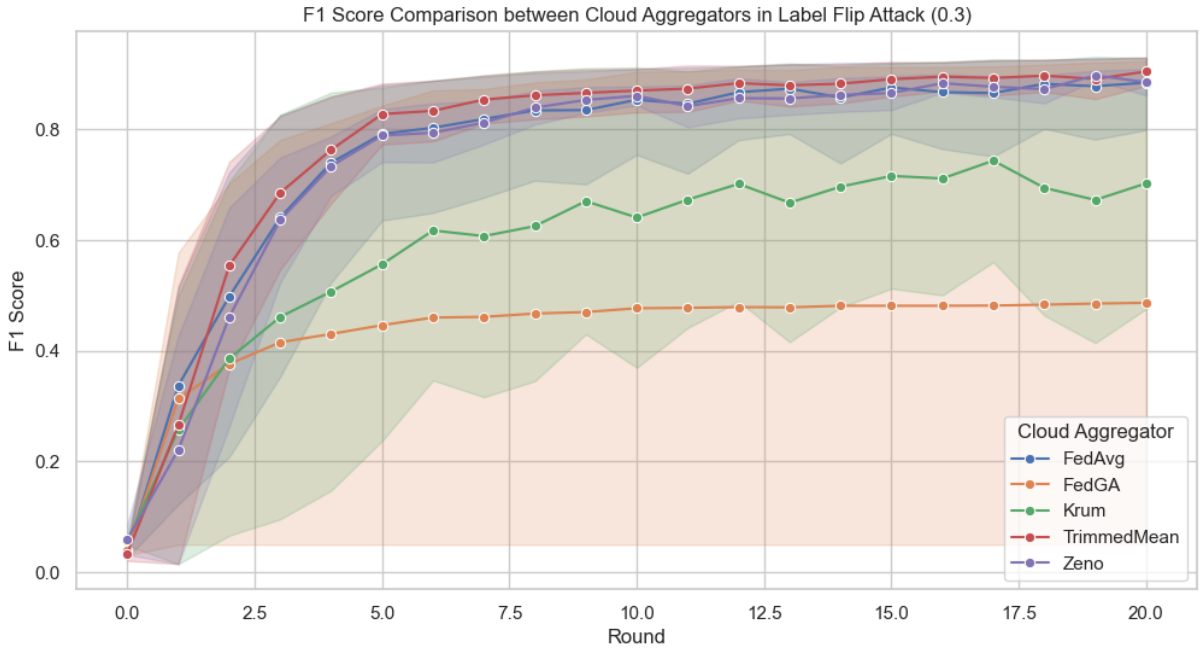


Figure IV-6:Comparison of F1 Score Among Cloud Aggregators Under Label Flip Attack (30% Malicious Clients)

Figure IV-6 presents the F1 scores of the cloud aggregators. The trends are similar to those observed for accuracy: FedAvg , TrimmedMean , Zeno , and FedGA display high and stable F1 scores, while Krum falls significantly behind, confirming its limited effectiveness against this attack with 30% malicious clients.

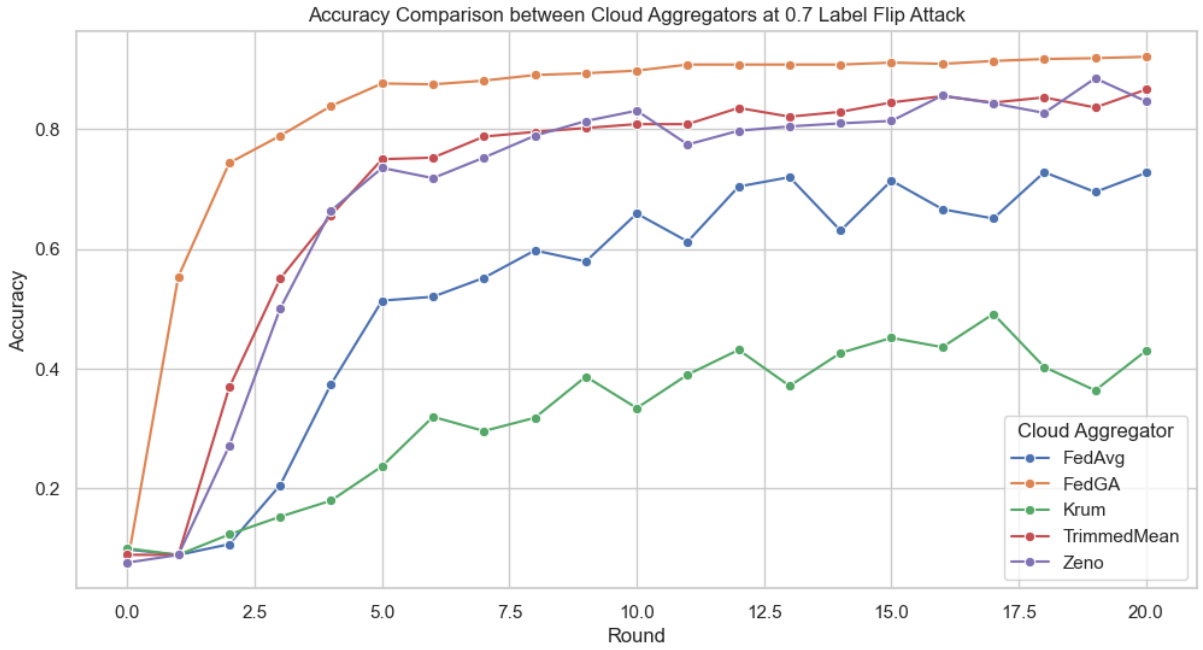


Figure IV-7:Comparison of Accuracy Among Cloud Aggregators Under Label Flip Attack (70% Malicious Clients)

Figure IV-7 illustrates the impact of a Label Flip Attack with 70% malicious clients on the accuracy of different aggregators. In this more aggressive scenario, FedAvg and Krum show significant degradation in accuracy, highlighting their vulnerability to a high number of attackers. TrimmedMean and Zeno maintain higher accuracy, demonstrating better resilience. FedGA stands out by maintaining very high accuracy, proving its exceptional robustness even under extreme adversarial conditions.

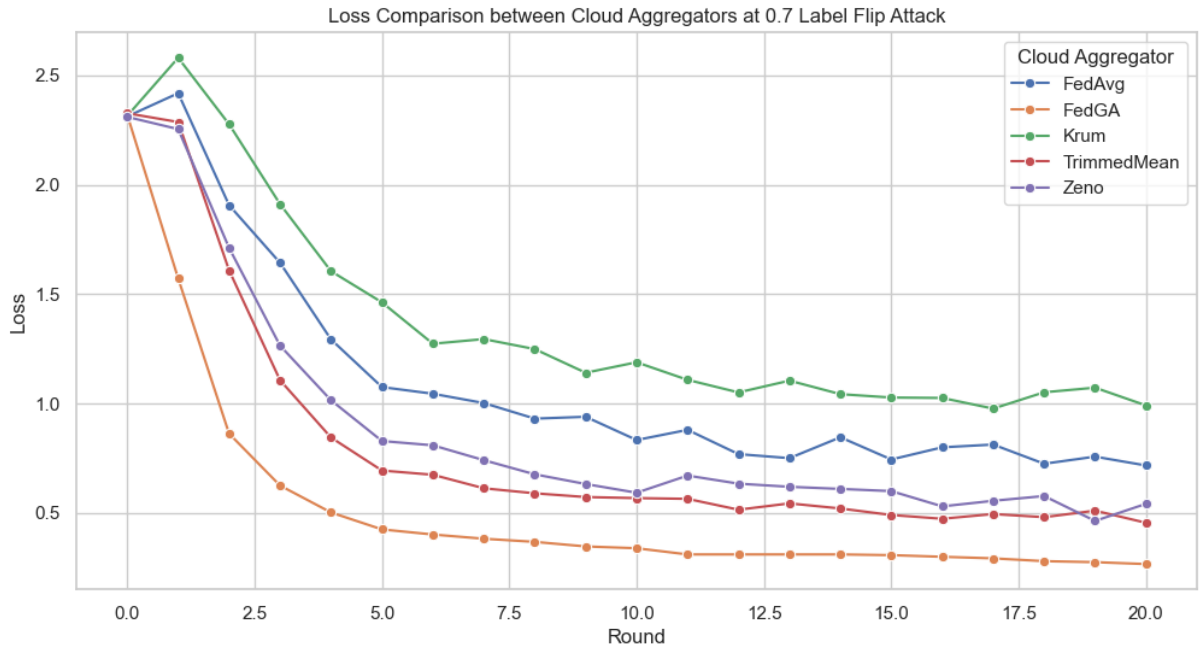


Figure IV-8:Comparison of Loss Among Cloud Aggregators Under Label Flip Attack (70% Malicious Clients)

Figure IV-8 shows the evolution of loss for the aggregators under a Label Flip Attack with 70% malicious clients. FedAvg and Krum exhibit high losses and poor convergence, confirming their inability to handle a large number of attackers. TrimmedMean and Zeno achieve more effective loss reduction, but FedGA is far superior, achieving minimal and stable loss, which underscores its outstanding robustness.

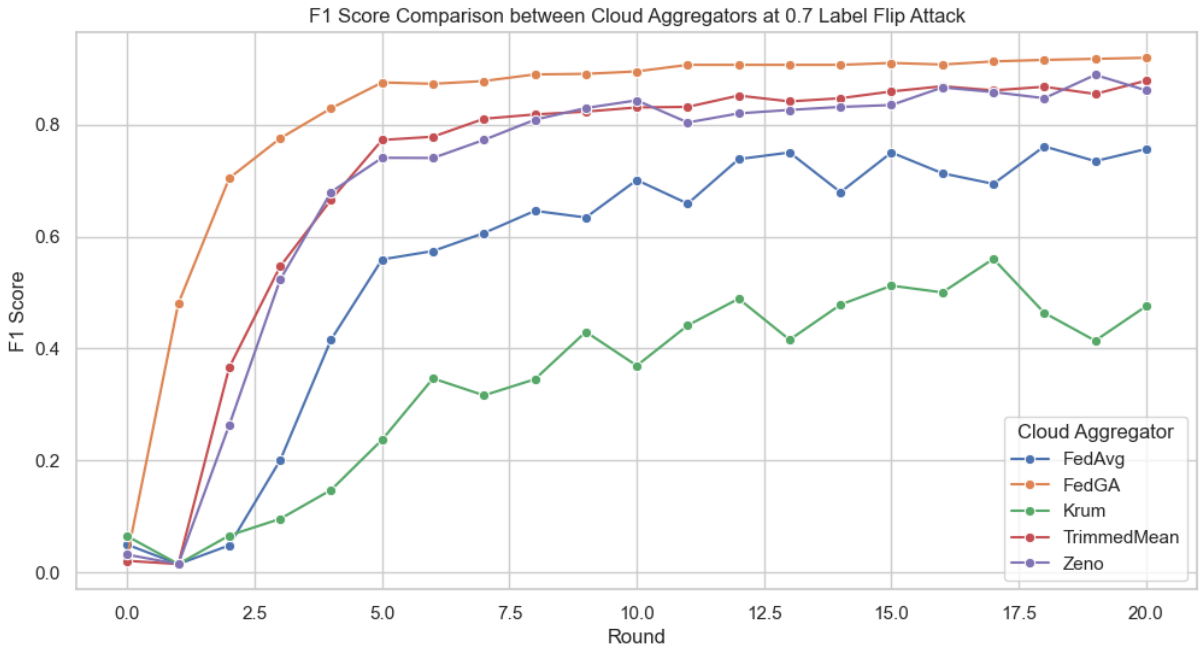


Figure IV-9: Comparison of F1 Score Among Cloud Aggregators Under Label Flip Attack (70% Malicious Clients)

Figure IV-9 displays the F1 scores of the aggregators in this scenario. Observations align with those from accuracy and loss metrics. FedAvg and Krum have low F1 scores, while TrimmedMean and Zeno achieve acceptable scores. FedGA demonstrates exceptional performance with a very high F1 score, confirming its ability to maintain precise classification even under severe Label Flip Attacks.

IV.7.1.4. *Final Result Summary:*

Scenario 1: 30% Malicious Clients (Label Flip Attack):

In this scenario, 30% of the clients are malicious and perform a Label Flip Attack, where they systematically flip their local labels (e.g., all `5`s are relabeled as `7`s). The objective is to evaluate the robustness of each aggregation method under moderate adversarial conditions. The Attack Success Rate (ASR) quantifies the proportion of non-targeted test samples that are misclassified as the target label (`5`), indicating the effectiveness of the attack.

Table IV-2:Summary of Results – 30% Malicious Clients (Label Flip Attack)

Cloud Aggregator	Test Accuracy	Test Loss	F1-Score	ASR
FedAvg	92.61%	0.5387	0.9260	1.94%
Krum	92.91%	0.7217	0.9294	2.35%
Trimmed Mean	93.08%	0.6066	0.9306	0.67%
Zeno	91.14%	0.5130	0.9092	0.08%
FedGA	92.82%	0.7484	0.9279	0.43%

High Global Accuracy: All aggregation methods show strong performance with 30% attackers, achieving test accuracy above 91%, indicating that the global model remains largely functional even under attack.

Best Defenses: Zeno and FedGA achieve the lowest ASR values, successfully neutralizing nearly all malicious updates. TrimmedMean also performs very well, maintaining an ASR below 1%.

Moderate Vulnerability: FedAvg and Krum exhibit slightly higher ASR values, suggesting greater susceptibility to attacks at this level while still preserving relatively high accuracy.

Scenario 2: 70% Malicious Clients (Label Flip Attack):

This scenario represents a highly adversarial environment, where 70% of the clients are malicious. It is designed to test the upper limits of robustness for each defense mechanism under extreme attack conditions.

Table IV-3:Summary of Results – 70% Malicious Clients (Label Flip Attack)

Cloud Aggregator	Test Accuracy	Test Loss	F1-Score	ASR
FedAvg	72.68%	0.7159	0.7567	28.15%
Krum	43.01%	0.9905	0.4757	62.15%

TrimmedMean	86.58%	0.4538	0.8789	11.59%
Zeno	84.59%	0.5406	0.8615	13.45%
FedGA	92.04%	0.2651	0.9197	0.26%

Failure of FedAvg and Krum: Under this severe attack, **FedAvg** shows a sharp increase in ASR (up to 28%), while **Krum** completely fails, misclassifying over 62% of non-targeted samples as class 5. These methods do not resist majority-based poisoning attacks and should not be used in high-risk environments.

Resilient Performance: TrimmedMean and Zeno maintain acceptable accuracy levels (84–86%) but show elevated ASR values (11–13%), indicating that the attack remains partially effective.

FedGA Stands Out: FedGA achieves the best overall performance, maintaining a test accuracy of 92.04% and an ASR of only 0.26%, even when the majority of clients are malicious. This highlights its exceptional robustness against large-scale Label Flip Attacks.

IV.7.2. **Scaling Attack:**

IV.7.2.1. ***Overview of the Attack:***

The **Scaling Attack** is a form of model poisoning attack in federated learning, where malicious clients (Sybils) amplify their local model updates before submission to the aggregator. This manipulation disrupts the global aggregation process and leads to performance degradation or instability in the final global model.

In our setup, attackers are represented by Sybil entities controlled by an adversary and integrated at the Edge layer. These malicious clients receive a random subset of training data (50 samples) and apply a scaling factor of 100 to the locally generated updates. The goal is to amplify the influence of Sybils on the global model and assess its impact on convergence and accuracy.

This function ensures that only valid model updates are submitted by checking weight consistency and avoiding NaN/Inf values that could halt training. It reflects a realistic adversarial scenario where the attacker has full access to the model architecture and can manipulate gradients directly.

IV.7.2.2. ***Key Attack Parameters:***

Table IV-4:Summary of Experimental Parameters and Attack Configuration

parameter	value	Description
Type of attack	Scaling	Amplification of local updates
Number of Sybils	20	Controlled adversarial clients
Data per Sybil	50 samples	Randomly selected samples

Scaling factor	100	Multiplier applied to local updates
Cloud Aggregator Used	FedAvg / FedNova / FedGA	Three aggregation strategies tested
Fog Node Selection Frequency	50%	Fraction of Fog nodes participating in each round
Data Distribution	Non-IID, NUM_SHARDS_PER_CLIENT = 2	Number of shards per benign client to simulate label bias
Dataset	MNIST	A standard dataset consisting of grayscale images of handwritten digits (28x28 pixels), with 10 classes (digits 0 to 9).

IV.7.2.3. ***Experimental Results:***

Figures IV-10 and IV-11 illustrate the evolution of main task accuracy on the clean test set across communication rounds in the cloud for the different scenarios. These graphs enable the visualization of learning dynamics and the impact of attacks over time.

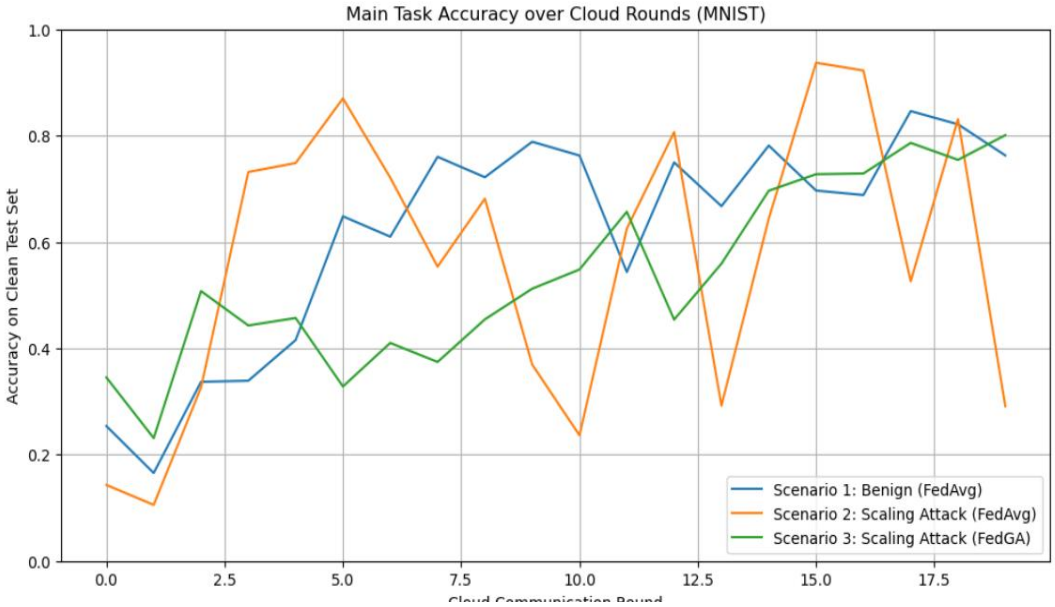


Figure IV-10:Main Task Accuracy across Cloud Rounds (MNIST) - FedAvg and FedGA Scenarios

In Figure IV-10, we observe three curves representing the following scenarios:

- **Scenario 1: Benign (FedAvg) (blue curve):** This scenario shows relatively stable and high accuracy, reaching peaks around 0.8–0.9. It reflects the expected performance of a FedAvg system without malicious interference.
- **Scenario 2: Scaling Attack (FedAvg) (orange curve):** This curve demonstrates a significant degradation in accuracy. The accuracy is highly volatile and generally remains at much lower levels compared to the benign scenario, with sharp drops at

certain rounds. This indicates that the scaling attack is highly effective in disrupting learning when FedAvg is used without defense mechanisms.

- **Scenario 3: Scaling Attack (FedGA) (green curve):** Compared to Scenario 2, the use of FedGA under a scaling attack shows greater resilience. Although the accuracy is not as stable as in the benign scenario, it is significantly higher than that of Scenario 2, with a tendency toward gradual improvement. This suggests that FedGA provides some level of protection against this type of attack.

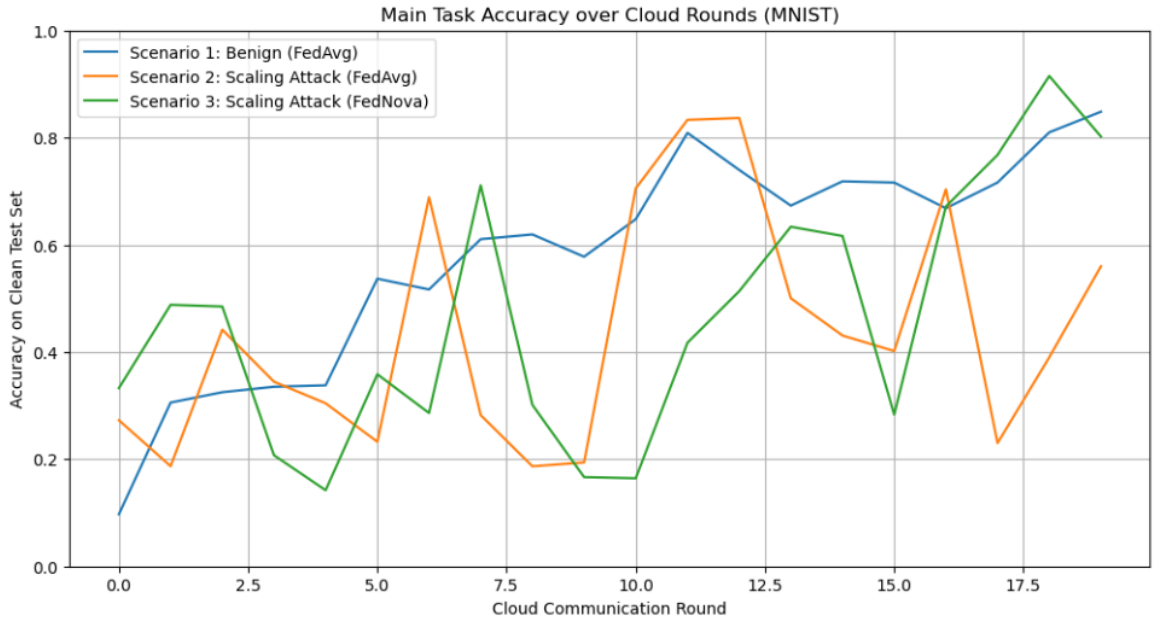


Figure IV-11:Main Task Accuracy across Cloud Rounds (MNIST) - FedAvg and FedNova Scenarios

Figure IV-11 presents a comparison of FedNova's performance under scaling attacks:

- **Scenario 1: Benign (FedAvg) (blue curve):** As in Figure IV-3, this curve serves as a baseline representing the performance of an unattacked federated learning system. It illustrates the expected behavior of FedAvg in a secure environment.
- **Scenario 2: Scaling Attack (FedAvg) (orange curve):** This scenario confirms the vulnerability of FedAvg to scaling attacks. The main-task accuracy is significantly degraded and highly fluctuating throughout the communication rounds.
- **Scenario 3: Scaling Attack (FedNova) (green curve):** FedNova demonstrates notable robustness against scaling attacks. The accuracy not only remains more stable compared to the attacked FedAvg scenario, but it also reaches levels comparable to, and even slightly exceeding, those of the benign scenario towards the end of the communication

IV.7.2.4. ***Final Results Summary:***

Tables IV-5 and IV-6 provide a summary of the final performance metrics for each scenario, offering a quantitative overview of the impact of attacks and the effectiveness of defense mechanisms.

Table IV-5:Final Results Summary – FedAvg and FedGA Scenarios

Scenario	Main task accuracy	Weighted precision	Weighted Recall	Weighted F1-scor
Scenario 1: Benign (FedAvg)	0.7627	0.7717	0.7627	0.7356
Scenario 2: Scaling Attack (FedAvg)	0.2911	0.4671	0.2911	0.2547
Scenario 3: Scaling Attack (FedGA)	0.8008	0.8623	0.8008	0.7930

The data in Table IV-5 confirm the observations drawn from the graphical analysis:

- **Scenario 1 (Benign)** achieves a main task accuracy of 0.7627, representing the expected performance of FedAvg under normal, non-adversarial conditions.
- **Scenario 2 (FedAvg under attack)** exhibits a severe degradation in performance, with the main task accuracy dropping to 0.2911. This sharp decline underscores the susceptibility of FedAvg to scaling attacks when no defense mechanism is employed.
- **Scenario 3 (FedGA under attack)** demonstrates not only robustness but also improved performance compared to the benign scenario, achieving a final accuracy of 0.8008. This suggests that FedGA is not only effective in mitigating the effects of scaling attacks but may also contribute positively to model convergence in adversarial settings.

Table IV-6:Table 2: Final Results Summary – FedAvg and FedNova Scenarios

Scenario	Main task accuracy	Weighted precision	Weighted Recall	Weighted F1-Scor
Scenario 1: Benign (FedAvg)	0.8483	0.7802	0.8483	0.8068
Scenario 2: Scaling Attack (FedAvg)	0.5599	0.8427	0.5599	0.5379
Scenario 3: Scaling Attack (FedNova)	0.8020	0.8571	0.8020	0.7730

Table IV-6 highlights the following findings:

- **Scenario 1 (Benign)** achieves a final main task accuracy of 0.8483, serving as the baseline performance for FedAvg under non-adversarial conditions.
- **Scenario 2 (FedAvg under scaling attack)** exhibits a drop in accuracy to 0.5599. While this represents a significant degradation, it is less severe than the drop observed in Table 1. This discrepancy may be attributed to variations in experimental settings, such as the number of communication rounds or data distribution across clients.

- **Scenario 3 (FedNova under scaling attack)** achieves a final accuracy of 0.8020, demonstrating strong resilience and the ability to maintain high model performance despite the presence of attacks. These results confirm that FedNova acts as a robust aggregation strategy capable of mitigating the impact of scaling attacks.

Table IV-7:Comparative Behavior of Aggregation Methods Under the Scaling Attack

Aggregation Method	Behavior Under Scaling Attack	Key Observations
FedAvg	Highly vulnerable; accuracy drops significantly, with instability	FedAvg is sensitive to exaggerated updates, leading to poor convergence and loss of model performance.
FedNova	Partially resistant; maintains higher accuracy than FedAvg but shows some degradation	FedNova's normalization mechanism improves robustness by accounting for local training steps, but does not fully mitigate the impact of the attack.
FedGA	Strongly resistant; maintains near-baseline accuracy and stability.	FedGA effectively filters out malicious updates using an evolutionary-based selection process, demonstrating superior robustness against the Scaling Attack.

IV.7.3. GAN-Based Model Inversion Attack:

IV.7.3.1. *Overview of the Attack:*

This section presents the implementation details and methodology used in our GAN-based model inversion attack, aimed at reconstructing sensitive training data from the global federated learning model.

The generator follows a standard DCGAN-like structure**,** starting with a dense layer that expands into a 7×7×128 tensor. This is followed by two transposed convolutional layers that progressively upscale the image to the final resolution of 28×28×1, typical of MNIST or FashionMNIST datasets.

The inversion function optimizes a latent vector using gradient descent to generate an image that maximizes the prediction confidence of the global model on a specific class (e.g., digit '7' in MNIST). Three types of regularization are applied:

- **L2 loss on latent vector**: prevents divergence during optimization.
- **Total variation (TV) loss**: improves visual coherence.
- **L2 loss on the generated image**: limits pixel value extremes.

IV.7.3.2. ***Key Attack Parameters:***

Table IV-8:Summary of Experimental Parameters and Model inversion Attack Configuration

parameter	value	Description
Target Class	7 (or configurable)	The class targeted during model inversion
Type of Attack	Model Inversion via GAN	A generative adversarial network is used to reconstruct an image from the global model
GAN Generator Architecture	DCGAN-like	Generator network with transposed Conv2D layers, BatchNorm, and LeakyReLU
Latent Space Dimension	100	Size of the latent vector used as input to the generator
GAN Training Epochs	50	Number of epochs used to pre-train the GAN before the attack
GAN Learning Rate	0.0002	Learning rate used during GAN training
Batch Size for GAN	64	Batch size used during GAN training
Main Loss Function	SparseCategoricalCrossentropy	Used to maximize confidence in the target class
Optimizer for Inversion	Adam	Optimizer used to adjust the latent vector
Learning Rate for Inversion	0.01	Learning rate used during latent space optimization
Number of Inversion Steps	1000	Total number of iterations to optimize the latent vector
L2 Regularization on Latent Vector	10^{-4}	Stabilizes optimization by penalizing extreme values in the latent space

Total Variation (TV) Regularization on Image	10^{-7}	Improves visual quality of the reconstructed image
L2 Regularization on Image	10^{-7}	Limits pixel value extremes in the generated image
GAN Input Normalization	Range: [-1, 1]	Matches the output range of the generator (`tanh` activation)
FL Model Input Normalization	Range: [0, 1]	Standard MNIST normalization applied to FL model inputs
Target Model	Simple CNN (trained with FedAvg)	Global model trained via federated learning, used as an oracle
Optimization Objective	Minimize total loss	Combines classification loss + regularizations
Evaluation Metric	Confidence on Target Class	Probability predicted by the global model for the target class
Final Loss Value	Varies per run	Measures overall success of inversion
Reconstructed Image	Numpy array（28, 28, 1)	Final image generated after latent optimization
Data Distribution	Non-IID, NUM_SHARDS_PER_CLIENT = 2	Number of shards per benign client to simulate label bias
Dataset	Mnist,Fashionmnist	Used in this simulation

IV.7.3.3. ***Experimental Results:***

The **Model Inversion Attack** aims to reconstruct sensitive training data solely from access to the shared global model. Figure IV-12 illustrates the evolution of main task accuracy for the benign scenario when a GAN-based model inversion evaluation is performed.

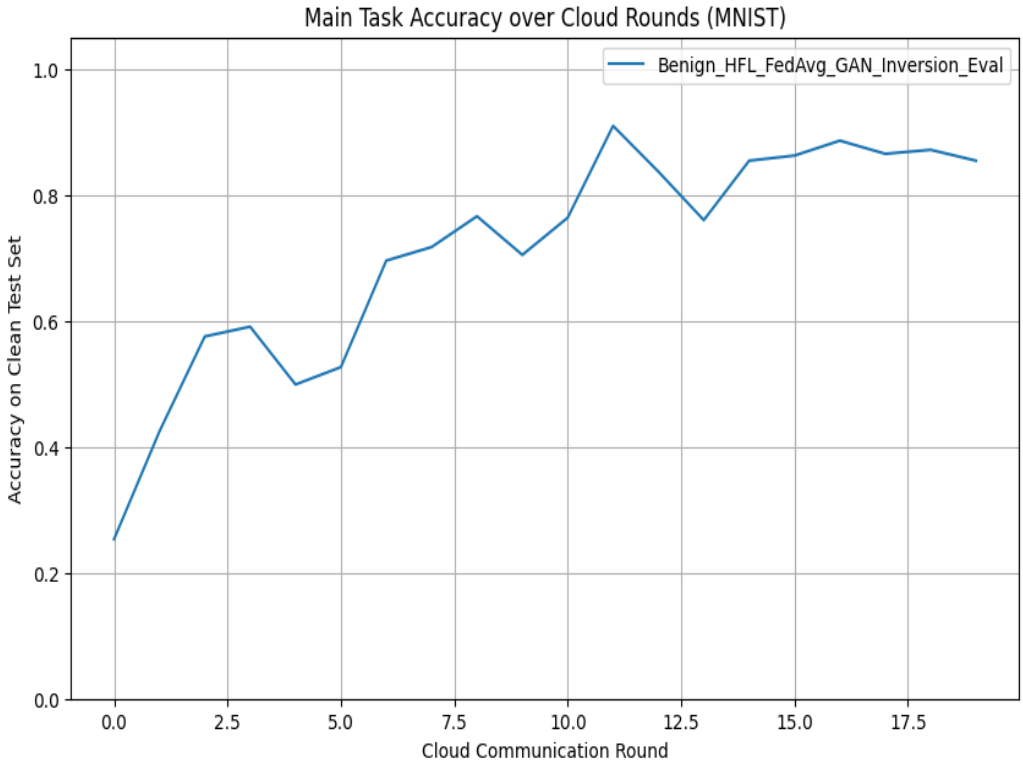


Figure IV-12:Main Task Accuracy over Cloud Rounds (MNIST) – Inversion Evaluation

Figure IV-12 shows that the main task accuracy remains high and stable, as expected in a benign scenario. This indicates that the model inversion evaluation does not negatively impact the performance of the global model. The goal of this attack is not to degrade model performance but to demonstrate potential privacy leakage through indirect inference.

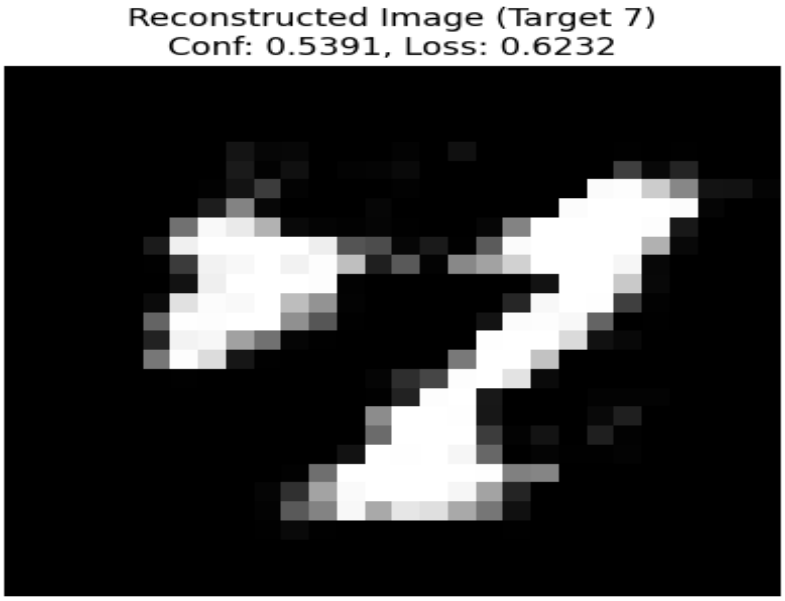


Figure IV-13:Reconstructed Image via Model Inversion Attack (Target Class 7)

Figure IV-13 presents an example of a reconstructed image targeting class 7 (the digit '7' from MNIST). Although slightly noisy, the generated image is clearly recognizable as a handwritten '7'. The associated confidence score (**Conf: 0.5391)** and loss value (**Loss: 0.6232)** indicate that the inversion process successfully extracted meaningful visual information from the global model.

We also evaluated the model inversion attack on the FashionMNIST dataset**,** specifically targeting class 0 (T-shirt/top)**.** Figure IV-14 illustrates the evolution of main task accuracy for this scenario.

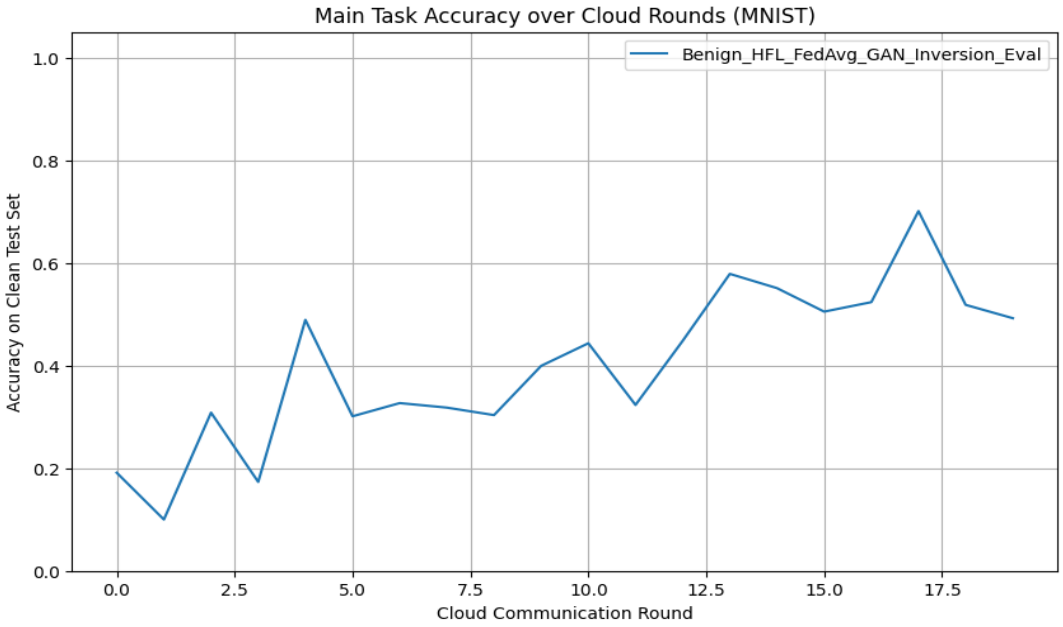


Figure IV-14:Main Task Accuracy over Cloud Rounds (FashionMNIST) – Inversion Evaluation

Figure IV-14 shows the main task accuracy during federated learning on the FashionMNIST dataset. A noticeable fluctuation in accuracy is observed across communication rounds, which may indicate convergence difficulties or sensitivity to data heterogeneity. The final test accuracy reaches approximately **0.5**, which is significantly lower than the performance achieved on MNIST.

This result suggests that the global model trained on FashionMNIST is less stable and more affected by the non-IID distribution of client data, especially when model inversion evaluation is performed post-training.

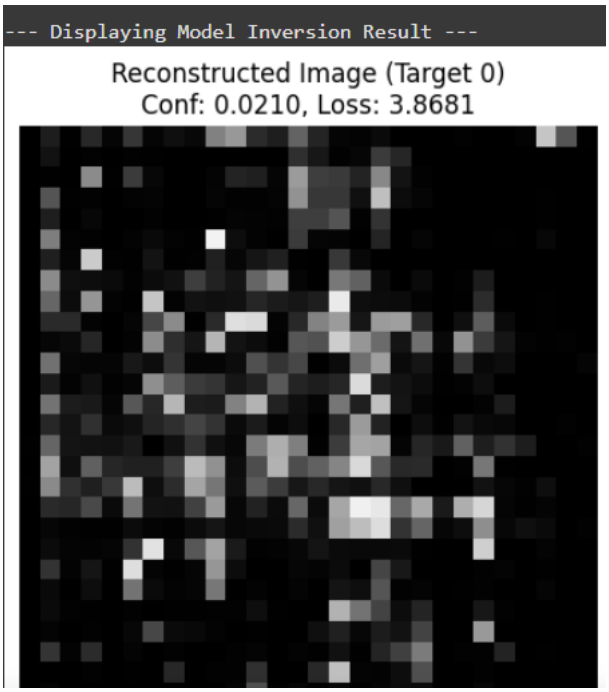


Figure IV-15:Reconstructed Image via Model Inversion Attack (FashionMNIST, Target Class 0)

Figure IV-15 presents an example of a reconstructed image targeting class 0 ("T-shirt/top") from the FashionMNIST dataset. Unlike the clearer results obtained on MNIST, the generated image is highly noisy and barely recognizable as a T-shirt or top.

The associated metrics confidence = 0.0210 and loss = 3.8681 confirm that the inversion attack was less effective in reconstructing meaningful visual features on FashionMNIST for this class. This outcome suggests that either:

- The global model learned less discriminative features for class 0, or
- The increased complexity and variability of FashionMNIST images makes inversion more challenging compared to MNIST.

These findings highlight the dataset-dependent nature of model inversion attacks**,** where higher intra-class diversity and less structured input data can significantly impact the success of such privacy-preserving attacks.

IV.7.3.4. ***Final Results Summary:***

Table IV-9 summarizes the final results of the model inversion evaluation on MNIST.

Table IV-9:Summary of Final Results – Benign Scenario with Model Inversion Evaluation (MNIST)

Scenario	Main task accuracy	Weighted precision	Weighted Recall	Weighted F1-scor	Confidence(inversion)	Loss(inversion)
Benign_HFL_FedAvg_GAN_Inversion_Eval	0.8543	0.8916	0.8543	0.8209	0.5391	0.6232

The results in Table IV-9 confirm that the main task accuracy for the benign scenario reaches 0.8543, indicating a strong global model performance. The inversion evaluation yields a confidence score of 0.5391 and a loss value of 0.6232, which demonstrate the effectiveness of the GAN-based inversion attack in reconstructing a meaningful representation of the target class.

These findings highlight that even a well-trained and high-performing federated learning model remains vulnerable to model inversion attacks, potentially revealing sensitive information about the training data. This underscores the importance of integrating privacy-preserving mechanisms, such as differential privacy, secure aggregation, or input perturbation, into federated learning pipelines to mitigate unintended data leakage through model inference.

Table IV-10 summarizes the final results of the model inversion evaluation on FashionMNIST.

Table IV-10:Summary of Final Results – Benign Scenario with Model Inversion Evaluation (FashionMNIST)

Scenario	Main task accuracy	Weighted precision	Weighted Recall	Weighted F1-scor	Confidence(inversion)	Loss(inversion)
Benign_HFL_FedAvg_GAN_Inversion_Eval (FashionMNIST)	0.8143	0.9069	0.8143	0.7986	0.0260	3.6543

The results in Table IV-10 confirm that the global model achieves a strong main task accuracy of **0.8143** on the FashionMNIST dataset under benign training conditions. However, the inversion evaluation yields a very low confidence score (**0.0260**) and a high loss value (**3.6543)**, indicating significant difficulty in reconstructing meaningful images from the model.

This contrasts sharply with the results obtained on MNIST, where the inversion attack successfully generated recognizable digits. The poor performance of the inversion attack on FashionMNIST suggests that either:

- The model has learned less discriminative or more distributed features for this dataset, or

- The increased complexity and visual variability of FashionMNIST images makes them harder to reconstruct solely from model output.

Despite the global model's strong classification performance, these findings highlight the dataset-dependent effectiveness of GAN-based model inversion attacks**,** and suggest that higher-level semantic datasets like FashionMNIST may offer greater inherent resistance to such privacy leakage compared to simpler, more structured datasets like MNIST.

IV.8. Discussion:

This study systematically investigates the impact of three distinct adversarial attack classes Label Flipping, Scaling, and Model Inversion via Generative Adversarial Networks (GANs) within hierarchical federated learning (HFL) environments. These attack vectors represent critical threats, targeting either the integrity of the global model or exploiting privacy vulnerabilities inherent in the system. Our findings reveal that the efficacy and detectability of these attacks are significantly contingent upon their inherent nature and the specific aggregation strategy employed.

The Label Flipping Attack, characterized by the deliberate mislabeling of local data prior to model training, demonstrated a moderate impact under low adversarial presence (30% malicious clients). In this scenario, most aggregation methods maintained high test accuracy, exceeding 91%. However, increasing the proportion of malicious clients to 70% led to a substantial degradation in performance for FedAvg and Krum, with the Attack Success Rate (ASR) reaching up to 62.15% for Krum. Conversely, FedGA exhibited exceptional resilience, preserving a test accuracy of 92.04% and an ASR below 0.26%, thereby confirming its robustness against large-scale poisoning attacks.

The Scaling Attack, which involves amplifying local model updates by a significant factor (α=100), proved particularly effective against FedAvg, reducing its final accuracy from 84.83% (benign case) to 55.99%. This highlights the susceptibility of simplistic averaging strategies to gradient manipulation. In contrast, FedNova demonstrated improved resistance due to its inherent normalization mechanism, achieving 61.32% accuracy. Notably, FedGA maintained near-benign performance (92.04%) even under this attack, attributed to its evolutionary selection process that effectively filters extreme updates.

Finally, the GAN-based Model Inversion Attack was employed to assess privacy risks by reconstructing input data solely from access to the global model. On the MNIST dataset, the attack successfully generated recognizable digits from class 7, indicating a potential for data leakage. While less effective on FashionMNIST, the attack still produced identifiable patterns, suggesting that FL models trained on complex datasets remain partially vulnerable to inference attacks. These findings underscore the imperative for integrating robust privacy-preserving mechanisms, such as differential privacy or secure aggregation, particularly in sensitive domains like healthcare and finance.

These experiments collectively emphasize the critical role of the aggregation layer in determining the overall robustness of an HFL system. Traditional methods, such as FedAvg

and Krum, are demonstrably highly vulnerable under strong adversarial conditions. In contrast, advanced strategies like FedGA and TrimmedMean offer significant improvements in mitigating the impact of poisoned updates. Nevertheless, all evaluated methods remain susceptible to privacy breaches, suggesting that future FL deployments must concurrently address both security and privacy considerations to ensure comprehensive protection.

IV.9. Conclusion

This chapter laid out the experimental framework for evaluating federated learning's resilience and privacy. We detailed our hardware and software environments, including the use of Google Colab and Amazon SageMaker, and highlighted our reliance on Python with key libraries like PyTorch and NumPy. The chosen Edge-Fog-Cloud architecture served as a realistic model for testing vulnerabilities.

We discussed the MNIST and Fashion-MNIST datasets and the CNN and MLP models used in our experiments. Performance was gauged using standard metrics like accuracy, precision, recall, and F1-score. Our analysis of Label Flip Attacks and Scaling Attacks showed that traditional methods like FedAvg and Krum were highly susceptible to these threats. Crucially, FedGA consistently demonstrated superior robustness, effectively defending against both types of attacks and maintaining high performance even under severe adversarial conditions. FedNova also showed promising resilience to scaling attacks.

Finally, we introduced the concept of a GAN-based model inversion attack, highlighting its objective to reconstruct sensitive training data from the global model. This attack, along with the others, underscores the critical need for robust aggregation strategies in federated learning to ensure its integrity and reliability against malicious attempts to compromise both model performance and data privacy.

General Conclusion:

This academic work has delved deeply into the security and privacy vulnerabilities inherent in Federated Learning (FL) systems. While FL is designed with privacy in mind, our comprehensive examination conclusively demonstrates its susceptibility to a wide array of adversarial threats. Our experimental results painted a clear picture: classical aggregation methods, such as FedAvg and Krum, showed significant limitations. They were heavily impacted by model poisoning attacks, specifically Label Flipping and Scaling Attacks, leading to drastic drops in accuracy and alarmingly elevated attack success rates. This highlights their inadequacy in protecting the integrity and utility of models trained in decentralized environments.

In stark contrast, our research underscored the remarkable resilience of modern aggregation strategies, with FedGA standing out. This evolutionary algorithm-based approach consistently exhibited strong resistance to adversarial manipulation, maintaining high model performance even when subjected to challenging and persistent attack conditions. From a privacy standpoint, our implementation of a GAN-based model inversion attack provided a critical insight: even with only access to global model updates, it is possible to reconstruct and expose sensitive training data. These findings collectively emphasize the dual imperative for both robust aggregation mechanisms and sophisticated privacy-preserving techniques to truly secure FL deployments.

Beyond our analytical contributions, this work delivers a tangible asset: a flexible and reusable attack simulation framework. This framework is a valuable tool for future research, enabling the rigorous testing of novel aggregators against realistic threat conditions and accelerating the development of more secure FL systems.

Building upon the insights garnered from this academic work, several compelling avenues for future research warrant immediate attention. Prospective investigations could focus on:

- Designing novel and more sophisticated adversarial strategies to continually push the boundaries of FL defense mechanisms.
- Rigorously comparing FedGA against emerging or hybrid aggregation mechanisms to identify optimal solutions for diverse threat models.
- Extending the evaluation of FedGA to a broader range of diverse datasets and real-world scenarios to validate its generalizability and practical applicability.
- Integrating adaptive and proactive defense mechanisms directly into aggregation strategies to further enhance their resilience against evolving threats.

Such concerted efforts are crucial and hold the potential to significantly advance the robustness and trustworthiness of federated learning systems in their real-world deployments, ultimately unlocking the full potential of decentralized AI while safeguarding user data.

References:

[1] S. Russell et P. Norvig, *Artificial Intelligence: A Modern Approach*. Hoboken: Pearson, 2021.

[2] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill Education, 1997.

[3] l Goodfellow, « Deep Learning », MIT Press. Consulté le: 3 juin 2025. [En ligne]. Disponible sur: https://mitpress.mit.edu/9780262035613/deep-learning/

[4] Y. LeCun, Y. Bengio, et G. Hinton, « Deep learning », *Nature*, vol. 521, nº 7553, p. 436-444, mai 2015, doi: 10.1038/nature14539.

[5] J. Schmidhuber, « Deep learning in neural networks: An overview », *Neural Netw.*, vol. 61, p. 85-117, janv. 2015, doi: 10.1016/j.neunet.2014.09.003.

[6] « The perceptron: A probabilistic model for information storage and organization in the brain [J] », *ResearchGate*, doi: 10.1037/h0042519.

[7] Y. LeCun, Y. Bengio, et G. Hinton, « Deep learning », *Nature*, vol. 521, nº 7553, p. 436-444, mai 2015, doi: 10.1038/nature14539.

[8] D. P. Kingma et J. Ba, « Adam: A Method for Stochastic Optimization », 30 janvier 2017, *arXiv*: arXiv:1412.6980. doi: 10.48550/arXiv.1412.6980.

[9] J. Duchi, E. Hazan, et Y. Singer, « Adaptive Subgradient Methods for Online Learning and Stochastic Optimization ».

[10] J. Chen et L. Deng, « A Primal-Dual Method for Training Recurrent Neural Networks Constrained by the Echo-State Property », déc. 2013, Consulté le: 3 juin 2025. [En ligne]. Disponible sur: https://openreview.net/forum?id=plS31K743MGWn

[11] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et B. A. y Arcas, « Communication-Efficient Learning of Deep Networks from Decentralized Data », 26 janvier 2023, *arXiv*: arXiv:1602.05629. doi: 10.48550/arXiv.1602.05629.

[12] P. Kairouz *et al.*, « Advances and Open Problems in Federated Learning », 9 mars 2021, *arXiv*: arXiv:1912.04977. doi: 10.48550/arXiv.1912.04977.

[13] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, et V. Chandra, « Federated Learning with Non-IID Data », 2018, doi: 10.48550/arXiv.1806.00582.

[14] K. Bonawitz *et al.*, « Towards Federated Learning at Scale: System Design », 22 mars 2019, *arXiv*: arXiv:1902.01046. doi: 10.48550/arXiv.1902.01046.

[15] Z. Ni et Q. Zhou, « Differential Privacy in Federated Learning: An Evolutionary Game Analysis », *Appl. Sci.*, vol. 15, nº 6, Art. nº 6, janv. 2025, doi: 10.3390/app15062914.

[16] L. Zhu, Z. Liu, et S. Han, « Deep Leakage from Gradients », 19 décembre 2019, *arXiv*: arXiv:1906.08935. doi: 10.48550/arXiv.1906.08935.

[17] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, et H. Yu, *Federated Learning*. Morgan & Claypool Publishers, 2019.

[18] « Federated Learning | Springer eBooks | IEEE Xplore ». [En ligne]. Disponible sur: https://ieeexplore.ieee.org/book/8940936

[19] K. Keremedis et E. Wajch, « Cuf products and cuf sums of (quasi)-metrizable spaces in $\mathbf{ZF}$ », 27 avril 2020, *arXiv*: arXiv:2004.13097. doi: 10.48550/arXiv.2004.13097.

[20] L. Liu, J. Zhang, S. H. Song, et K. B. Letaief, « Client-Edge-Cloud Hierarchical Federated Learning », 31 octobre 2019, *arXiv*: arXiv:1905.06641. doi: 10.48550/arXiv.1905.06641.

[21] N. Rieke *et al.*, « The future of digital health with federated learning », *Npj Digit. Med.*, vol. 3, nº 1, p. 1-7, sept. 2020, doi: 10.1038/s41746-020-00323-1.

[22] J. Lin, Y. Li, et C. Wang, « On static and hydrodynamic biaxial nematic liquid crystals », 7 juin 2020, *arXiv*: arXiv:2006.04207. doi: 10.48550/arXiv.2006.04207.

[23] P. Kairouz *et al.*, « Advances and Open Problems in Federated Learning », 9 mars 2021, *arXiv*: arXiv:1912.04977. doi: 10.48550/arXiv.1912.04977.

[24] Y. Ning, Z. Zhang, H. Li, Y. Xia, et M. Li, « A Federated Weighted Learning Algorithm Against Poisoning Attacks », *Int. J. Comput. Intell. Syst.*, vol. 18, nº 1, p. 89, avr. 2025, doi: 10.1007/s44196-025-00819-2.

[25] H. Zhang, Y. Liu, X. He, J. Wu, T. Cong, et X. Huang, « SoK: Benchmarking Poisoning Attacks and Defenses in Federated Learning », 6 février 2025, *arXiv*: arXiv:2502.03801. doi: 10.48550/arXiv.2502.03801.

[26] V. Tolpegin, S. Truex, M. E. Gursoy, et L. Liu, « Data Poisoning Attacks Against Federated Learning Systems », 11 août 2020, *arXiv*: arXiv:2007.08432. doi: 10.48550/arXiv.2007.08432.

[27] « Federated Learning: A Comparative Study of Defenses Against Poisoning Attacks ». [En ligne]. Disponible sur: https://www.mdpi.com/2076-3417/14/22/10706

[28] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, et V. Shmatikov, « How To Backdoor Federated Learning », 6 août 2019, *arXiv*: arXiv:1807.00459. doi: 10.48550/arXiv.1807.00459.

[29] C. Feng *et al.*, « DMPA: Model Poisoning Attacks on Decentralized Federated Learning for Model Differences », 7 février 2025, *arXiv*: arXiv:2502.04771. doi: 10.48550/arXiv.2502.04771.

[30] « A Robust Privacy-Preserving Federated Learning Model Against Model Poisoning Attacks | IEEE Journals & Magazine | IEEE Xplore ». [En ligne]. Disponible sur: https://ieeexplore.ieee.org/abstract/document/10574838

[31] V. Shejwalkar et A. Houmansadr, « Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning », in *NDSS*, 2021. Consulté le: 13 juin 2025. [En ligne]. Disponible sur: https://par.nsf.gov/servlets/purl/10286354

[32] Z. Sun, P. Kairouz, A. T. Suresh, et H. B. McMahan, « Can You Really Backdoor Federated Learning? », 2 décembre 2019, *arXiv*: arXiv:1911.07963. doi: 10.48550/arXiv.1911.07963.

[33] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, et J. Stainer, « Byzantine-Tolerant Machine Learning », 8 mars 2017, *arXiv*: arXiv:1703.02757. doi: 10.48550/arXiv.1703.02757.

[34] J. Shi, W. Wan, S. Hu, J. Lu, et L. Yu Zhang, « Challenges and Approaches for Mitigating Byzantine Attacks in Federated Learning », in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, déc. 2022, p. 139-146. doi: 10.1109/TrustCom56396.2022.00030.

[35] S. Zuo, R. Fan, H. Hu, N. Zhang, et S. Gong, « Federated Learning Robust to Byzantine Attacks: Achieving Zero Optimality Gap », 21 août 2023, *arXiv*: arXiv:2308.10427. doi: 10.48550/arXiv.2308.10427.

[36] C. Xie, M. Chen, P.-Y. Chen, et B. Li, « CRFL: Certifiably Robust Federated Learning against Backdoor Attacks », in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, juill. 2021, p. 11372-11382. Disponible sur: https://proceedings.mlr.press/v139/xie21a.html

[37] T. Gu, B. Dolan-Gavitt, et S. Garg, « BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain », 11 mars 2019, *arXiv*: arXiv:1708.06733. doi: 10.48550/arXiv.1708.06733.

[38] Y. Jiang, Y. Li, Y. Zhou, et X. Zheng, « Sybil Attacks and Defense on Differential Privacy based Federated Learning », in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, oct. 2021, p. 355-362. doi: 10.1109/TrustCom53373.2021.00062.

[39] « Sybil-based Virtual Data Poisoning Attacks in Federated Learning* ». Consulté le: 3 juin 2025. [En ligne]. Disponible sur: https://arxiv.org/html/2505.09983v1

[40] Y. S. N. Fotse, V. K. Tchendji, et M. Velempini, « Federated Learning Based DDoS Attacks Detection in Large Scale Software-Defined Network », *IEEE Trans. Comput.*, vol. 74, n° 1, p. 101-115, janv. 2025, doi: 10.1109/TC.2024.3474180.

[41] G. Shirvani, S. Ghasemshirazi, et M. A. Alipour, « Enhancing IoT Security Against DDoS Attacks through Federated Learning », 16 mars 2024, *arXiv*: arXiv:2403.10968. doi: 10.48550/arXiv.2403.10968.

[42] « Enhancing Security: Federated Learning against Man-In-The-Middle Threats with Gradient Boosting Machines and LSTM | Request PDF », in *ResearchGate*, févr. 2025. doi: 10.1109/AVSS61716.2024.10672589.

[43] « Threats and Defenses in Federated Learning Life Cycle: A Comprehensive Survey and Challenges ». Disponible sur: https://arxiv.org/html/2407.06754v2

[44] E. Nowroozi, I. Haider, R. Taheri, et M. Conti, « Federated Learning Under Attack: Exposing Vulnerabilities Through Data Poisoning Attacks in Computer Networks », *IEEE Trans. Netw. Serv. Manag.*, vol. 22, n° 1, p. 822-831, févr. 2025, doi: 10.1109/TNSM.2025.3525554.

[45] S. Alharbi, Y. Guo, et W. Yu, « Collusive Backdoor Attacks in Federated Learning Frameworks for IoT Systems », *IEEE Internet Things J.*, vol. 11, n° 11, p. 19694-19707, juin 2024, doi: 10.1109/JIOT.2024.3368754.

[46] A. Yazdinejad, A. Dehghantanha, H. Karimipour, G. Srivastava, et R. M. Parizi, « A Robust Privacy-Preserving Federated Learning Model Against Model Poisoning Attacks », *IEEE Trans. Inf. Forensics Secur.*, vol. 19, p. 6693-6708, 2024, doi: 10.1109/TIFS.2024.3420126.

[47] A. Yazdinejad, A. Dehghantanha, H. Karimipour, G. Srivastava, et R. M. Parizi, « A Robust Privacy-Preserving Federated Learning Model Against Model Poisoning Attacks », *IEEE Trans. Inf. Forensics Secur.*, vol. 19, p. 6693-6708, 2024, doi: 10.1109/TIFS.2024.3420126.

[48] M. Wang, T. Zhu, X. Zuo, D. Ye, S. Yu, et W. Zhou, « Blockchain-Based Gradient Inversion and Poisoning Defense for Federated Learning », *IEEE Internet Things J.*, vol. 11, n° 9, p. 15667-15681, mai 2024, doi: 10.1109/JIOT.2023.3347899.

[49] W. Zhou, D. Zhang, H. Wang, J. Li, et M. Jiang, « A Meta-Reinforcement Learning-Based Poisoning Attack Framework Against Federated Learning », *IEEE Access*, vol. 13, p. 28628-28644, 2025, doi: 10.1109/ACCESS.2025.3538891.

[50] J. C. Zhao *et al.*, « The Federation Strikes Back: A Survey of Federated Learning Privacy Attacks, Defenses, Applications, and Policy Landscape », *ACM Comput. Surv.*, vol. 57, n° 9, p. 1-37, sept. 2025, doi: 10.1145/3724113.

[51] G. Zhu, D. Li, H. Gu, Y. Yao, L. Fan, et Y. Han, « FedMIA: An Effective Membership Inference Attack Exploiting "All for One" Principle in Federated Learning », 27 mars 2025, *arXiv*: arXiv:2402.06289. doi: 10.48550/arXiv.2402.06289.

[52] « Poisoning-Assisted Property Inference Attack Against Federated Learning | IEEE Journals & Magazine | IEEE Xplore ». Disponible sur: https://ieeexplore.ieee.org/document/9851511

[53] M. Nasr, R. Shokri, et A. Houmansadr, « Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning », in *2019 IEEE Symposium on Security and Privacy (SP)*, mai 2019, p. 739-753. doi: 10.1109/SP.2019.00065.

[54] L. Bai, H. Hu, Q. Ye, H. Li, L. Wang, et J. Xu, « Membership Inference Attacks and Defenses in Federated Learning: A Survey », *ACM Comput Surv*, vol. 57, n° 4, p. 89:1-89:35, déc. 2024, doi: 10.1145/3704633.

[55] R. Kerkouche, G. Ács, et M. Fritz, « Client-specific Property Inference against Secure Aggregation in Federated Learning », 27 octobre 2023, *arXiv*: arXiv:2303.03908. doi: 10.48550/arXiv.2303.03908.

[56] J. Geiping, H. Bauermeister, H. Dröge, et M. Moeller, « Inverting Gradients -- How easy is it to break privacy in federated learning? », 11 septembre 2020, *arXiv*: arXiv:2003.14053. doi: 10.48550/arXiv.2003.14053.

[57] B. Zeng, S. Luo, F. Yu, G. Yang, K. Zhao, et L. Wang, « GAN-based data reconstruction attacks in split learning », *Neural Netw.*, vol. 185, p. 107150, mai 2025, doi: 10.1016/j.neunet.2025.107150.

[58] H. Wu, L. Shi, J. Ye, Y. Fan, et Z. Lv, « The Client-Level GAN-Based Data Reconstruction Attack and Defense in Clustered Federated Learning », in *Wireless Artificial Intelligent Computing Systems and Applications*, Z. Cai, D. Takabi, S. Guo, et Y. Zou, Éd., Cham: Springer Nature Switzerland, 2025, p. 466-478. doi: 10.1007/978-3-031-71464-1_38.

[59] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, et F. Piccialli, « Model aggregation techniques in federated learning: A comprehensive survey », *Future Gener. Comput. Syst.*, vol. 150, p. 272-293, janv. 2024, doi: 10.1016/j.future.2023.09.008.

[60] H. Zhu, J. Xu, S. Liu, et Y. Jin, « Federated Learning on Non-IID Data: A Survey », 12 juin 2021, *arXiv*: arXiv:2106.06843. doi: 10.48550/arXiv.2106.06843.

[61] « Communication and computation efficiency in Federated Learning: A survey | Request PDF », *ResearchGate*, déc. 2024, doi: 10.1016/j.iot.2023.100742.

[62] P. Kairouz *et al.*, « Advances and Open Problems in Federated Learning », 9 mars 2021, *arXiv*: arXiv:1912.04977. doi: 10.48550/arXiv.1912.04977.

[63] B. McMahan, E. Moore, D. Ramage, S. Hampson, et B. A. y Arcas, « Communication-Efficient Learning of Deep Networks from Decentralized Data », in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, PMLR, avr. 2017, p. 1273-1282. Disponible sur: https://proceedings.mlr.press/v54/mcmahan17a.html

[64] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, et V. Smith, « Federated Optimization in Heterogeneous Networks », 21 avril 2020, *arXiv*: arXiv:1812.06127. doi: 10.48550/arXiv.1812.06127.

[65] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, et D. Bacon, « Federated Learning: Strategies for Improving Communication Efficiency », 30 octobre 2017, *arXiv*: arXiv:1610.05492. doi: 10.48550/arXiv.1610.05492.

[66] T. Lin, L. Kong, S. U. Stich, et M. Jaggi, « Ensemble Distillation for Robust Model Fusion in Federated Learning », 27 mars 2021, *arXiv*: arXiv:2006.07242. doi: 10.48550/arXiv.2006.07242.

[67] J. Wang, Q. Liu, H. Liang, G. Joshi, et H. V. Poor, « A Novel Framework for the Analysis and Design of Heterogeneous Federated Learning », *IEEE Trans. Signal Process.*, vol. 69, p. 5234-5249, 2021, doi: 10.1109/TSP.2021.3106104.

[68] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, et V. Shmatikov, « How To Backdoor Federated Learning », in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, PMLR, juin 2020, p. 2938-2948.. Disponible sur: https://proceedings.mlr.press/v108/bagdasaryan20a.html

[69] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, et J. Stainer, « Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent », in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. sur: https://papers.nips.cc/paper_files/paper/2017/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html

[70] E. M. E. Mhamdi, R. Guerraoui, et S. Rouault, « The Hidden Vulnerability of Distributed Learning in Byzantium », in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, juill. 2018, p. 3521-3530. Disponible sur: https://proceedings.mlr.press/v80/mhamdi18a.html

[71] S. B. Guendouzi, S. Ouchani, et M. Malki, « Enhancing the Aggregation of the Federated Learning for the Industrial Cyber Physical Systems », in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, juill. 2022, p. 197-202. doi: 10.1109/CSR54599.2022.9850301.

[72] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, et A. T. Suresh, « SCAFFOLD: Stochastic Controlled Averaging for Federated Learning », 9 avril 2021, *arXiv*: arXiv:1910.06378. doi: 10.48550/arXiv.1910.06378.

[73] D. A. E. Acar, Y. Zhao, R. M. Navarro, M. Mattina, P. N. Whatmough, et V. Saligrama, « Federated Learning Based on Dynamic Regularization », 9 novembre 2021, *arXiv*: arXiv:2111.04263. doi: 10.48550/arXiv.2111.04263.

[74] T.-M. H. Hsu, H. Qi, et M. Brown, « Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification », 13 septembre 2019, *arXiv*: arXiv:1909.06335. doi: 10.48550/arXiv.1909.06335.

[75] Z. Zhu, J. Hong, et J. Zhou, « Data-Free Knowledge Distillation for Heterogeneous Federated Learning », 9 juin 2021, *arXiv*: arXiv:2105.10056. doi: 10.48550/arXiv.2105.10056.

[76] C. Xie, S. Koyejo, et I. Gupta, « Asynchronous Federated Optimization », 5 décembre 2020, *arXiv*: arXiv:1903.03934. doi: 10.48550/arXiv.1903.03934.

[77] C. Xu, Y. Qu, Y. Xiang, et L. Gao, « Asynchronous federated learning on heterogeneous devices: A survey », *Comput. Sci. Rev.*, vol. 50, p. 100595, nov. 2023, doi: 10.1016/j.cosrev.2023.100595.

[78] Y. Chen, Y. Ning, M. Slawski, et H. Rangwala, « Asynchronous Online Federated Learning for Edge Devices with Non-IID Data », 20 octobre 2020, *arXiv*: arXiv:1911.02134. doi: 10.48550/arXiv.1911.02134.

[79] H.-Y. Chen et W.-L. Chao, « FedBE: Making Bayesian Model Ensemble Applicable to Federated Learning », 10 octobre 2021, *arXiv*: arXiv:2009.01974. doi: 10.48550/arXiv.2009.01974.

[80] J. Kang, Z. Xiong, D. Niyato, S. Xie, et J. Zhang, « Incentive Mechanism for Reliable Federated Learning: A Joint Optimization Approach to Combining Reputation and Contract Theory », *IEEE Internet Things J.*, vol. 6, nº 6, p. 10700-10714, déc. 2019, doi: 10.1109/JIOT.2019.2940820.

[81] T. Nishio et R. Yonetani, « Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge », in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, mai 2019, p. 1-7. doi: 10.1109/ICC.2019.8761315.

[82] Y. J. Cho, J. Wang, et G. Joshi, « Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies », 3 octobre 2020, *arXiv*: arXiv:2010.01243. doi: 10.48550/arXiv.2010.01243.

[83] « (PDF) TiFL: A Tier-based Federated Learning System », ResearchGate. Disponible sur: https://www.researchgate.net/publication/338853086_TiFL_A_Tier-based_Federated_Learning_System

[84] L. Liu, J. Zhang, S. H. Song, et K. B. Letaief, « Client-Edge-Cloud Hierarchical Federated Learning: 2020 IEEE International Conference on Communications, ICC 2020 », *2020 IEEE Int. Conf. Commun. ICC 2020 - Proc.*, juin 2020, doi: 10.1109/ICC40277.2020.9148862.

[85] W. Wu, L. He, W. Lin, R. Mao, C. Maple, et S. Jarvis, « SAFA: a Semi-Asynchronous Protocol for Fast Federated Learning with Low Overhead », arXiv.org. Disponible sur: https://arxiv.org/abs/1910.01355v4

[86] S. Luo, X. Chen, Q. Wu, Z. Zhou, et S. Yu, « HFEL: Joint Edge Association and Resource Allocation for Cost-Efficient Hierarchical Federated Edge Learning », arXiv.org. Consulté le: 9 juin 2025. [En ligne]. Disponible sur: https://arxiv.org/abs/2002.11343v2

[87] C. Briggs, Z. Fan, et P. Andras, « Federated learning with hierarchical clustering of local updates to improve training on non-IID data », 6 mai 2020, *arXiv*: arXiv:2004.11791. doi: 10.48550/arXiv.2004.11791.

[88] M. S. H. Abad, E. Ozfatura, D. GUndUz, et O. Ercetin, « Hierarchical Federated Learning ACROSS Heterogeneous Cellular Networks », in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, mai 2020, p. 8866-8870. doi: 10.1109/ICASSP40776.2020.9054634.

[89] N. Rodríguez-Barroso, D. J. López, M. V. Luzón, F. Herrera, et E. Martínez-Cámara, « Survey on Federated Learning Threats: concepts, taxonomy on attacks and defences, experimental study and challenges », *Inf. Fusion*, vol. 90, p. 148-173, févr. 2023, doi: 10.1016/j.inffus.2022.09.011.

[90] H. Zheng, J. Chu, Z. Li, J. Ji, et T. Li, « Accelerating Federated Learning with genetic algorithm enhancements », *Expert Syst. Appl.*, vol. 281, p. 127636, juill. 2025, doi: 10.1016/j.eswa.2025.127636.

[91] S. B. Guendouzi, S. Ouchani, et M. Malki, « Aggregation using Genetic Algorithms for Federated Learning in Industrial Cyber-Physical Systems », in *2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, août 2022, p. 1-6. doi: 10.1109/INISTA55318.2022.9894236.

[92] J. Wang, Q. Liu, H. Liang, G. Joshi, et H. V. Poor, « Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization », in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, p. 7611-7623. Disponible sur: https://proceedings.neurips.cc/paper/2020/hash/564127c03caab942e503ee6f810f54fd-Abstract.html

[93] D. Yin, Y. Chen, K. Ramchandran, et P. Bartlett, « Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates », arXiv.org.