DESIGN AND IMPLEMENTATION OF FEDERATED LEARNING FOR PRIVACY PRESERVING MACHINE LEARNING

BY

EMUZE OSAIGBOVO GLORY

PSC2105327

DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF PHYSICAL SCIENCES,

UNIVERSITY OF BENIN,

BENIN CITY,

EDO STATE, NIGERIA.

NOVEMBER 2025

DESIGN AND IMPLEMENTATION OF FEDERATED LEARNING FOR PRIVACY PRESERVING MACHINE LEARNING

BY

EMUZE OSAIGBOVO GLORY

PSC2105327

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF PHYSICAL SCIENCES, UNIVERSITY OF BENIN, BENIN CITY

IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF A BACHELOR OF SCIENCE (B.Sc.) DEGREE IN COMPUTER SCIENCE

NOVERMBER 2025

CERTIFICATION

This is to certify that this project work was carried out by **EMUZE OSAIGBOVO GLORY** with Matriculation Number **PSC2105327** under my supervision. It is adequate and satisfactory, both in scope and content, for the award of Bachelor of Science (B.sc) Degree in Computer Science of the University of Benin

DR. (MRS.) A.R. USIOBAIFO
Project Supervisor

DATE

APPROVAL

This project work is hereby approved in partial fulfilment of the requirements for the award of Bachelor of Science (B.Sc.) Degree in Computer Science from the University of Benin.

DR. (MRS.) A.R. USIOBAIFO
Head of Department

DATE

DEDICATION

This project is dedicated to God Almighty for giving me the strength and wisdom to see it through to completion, and even throughout my stay in the University of Benin (UNIBEN). It is also dedicated to my parents; Mr and Mrs Emuze and my uncles Mr Douglas Edoma and Mr Wlison Edoma; for their love, support and guidance throughout my academic journey.

ACKNOWLEDGEMENT

My utmost acknowledgement goes to God Almighty for giving me the strength, wisdom and direction throughout my academic journey. I would like to express my gratitude to my project supervisor who is also the Head of the Department Of Computer Science, Dr. (Mrs.) A.R. Usiobaifo for her consistent guidance towards ensuring the successful completion of this project.

I would also like to specially thank my project coordinator Dr. (Mrs.) A.R. Usiobaifo, and other lecturers in the Department of Computer Science who I have been opportune to cross paths with, and have impacted me immensely these past few years.

Finally, I also want to appreciate those who contributed to the success of this project: Emma Edoma, Eseosa, Bro Osasu. I would also like to thank my family and friends for their support, words of encouragement, and consistent guidance throughout this project.

Table of Contents

ABSTRACT

Machine learning has become an important technology used in many areas such as healthcare, banking, education, and mobile applications. However, traditional machine learning methods usually require large amounts of data to be collected and stored in one central location, which can create privacy and security concerns. This project focuses on the design and implementation of a Federated Learning system for privacy-preserving machine learning.

Federated Learning is a modern approach that allows multiple users or organizations to train a machine learning model together without sharing their raw data. Instead of sending personal or sensitive information to a central server, each participant trains the model on their local device and only shares model updates. This helps to protect privacy while still allowing the model to learn effectively.

In this project, a federated learning prototype was developed using Python and machine learning tools. The system was designed with features such as secure aggregation and differential privacy to improve data protection and reduce the risk of information leakage. The performance of the system was tested using distributed datasets to simulate real-world situations where data is stored in different locations.

The results showed that the proposed system was able to maintain a good level of model accuracy while protecting user privacy. It also demonstrated that federated learning can be a practical solution for organizations that need to use machine learning without exposing sensitive data. The study concludes that federated learning has great potential for institutions such as the University of Benin and other organizations in Nigeria that require secure and privacy-aware machine learning solutions.

CHAPTER ONE

INTRODUCTION

1.0 Background

Machine learning (ML) and artificial intelligence (AI) have moved from niche research topics into tools used in everyday services from smart keyboard suggestions and voice assistants to automated reading of medical images and fraud detection in banking. These systems improve as they are exposed to larger and more varied datasets. Traditionally, organizations collect data centrally and train models on that pooled data (centralized learning). While straightforward, this approach creates legal and security problems: sensitive records are harder to protect and many jurisdictions restrict how personal data may be shared.

Federated learning (FL) reverses that pattern by bringing the model to the data instead of collecting the data in one place. A server distributes a model to multiple clients (such as hospitals, banks, or personal devices). Each client updates the model locally using its private data and sends only the updates (model weight changes) back to the server. The server aggregates the updates to produce an improved global model raw data never leave local control. FL therefore reduces the need to pool sensitive records and enables joint training across institutions without exchanging underlying data. Large firms already use FL for some consumer features, and it holds clear promise for collaborative applications in healthcare, finance, and academia.

1.1 Motivation

Two practical forces drive this research: the need to use distributed but non-shareable data, and growing expectations for privacy. Many organizations and research groups maintain valuable datasets that cannot be consolidated due to policy, legal, or ethical constraints. FL offers a way to

learn from these dispersed data sources while respecting those limits. At the same time, individuals and regulators demand stronger controls over personal information; keeping raw data local aligns with data-minimization principles and helps preserve trust. For universities such as UNIBEN, FL could enable inter-departmental or university–hospital collaborations that improve models without exposing patient or student records.

Beyond policy and trust, concrete use cases exist: hospitals can collaboratively train diagnostic models to detect rare conditions, banks can improve fraud detection by combining signals across institutions, and academic researchers can share model improvements without sharing raw datasets. This project aims to create a practical FL prototype and evaluation that speaks directly to those opportunities in the Nigerian context.

1.2 Statement of Problem

While Federated Learning (FL) fundamentally solves the problem of data sharing, its real-world implementation introduces critical challenges regarding trustworthiness and performance.

For an institution like the University of Benin to safely collaborate on sensitive data (e.g., health or student records), the FL system must be engineered to withstand realistic threats.

We focus on solving five specific, practical problems confirmed by the project chapters:

1. Data Leakage and Confidentiality: Model updates, even without raw data, can be used by attackers or the central server to infer private details.

2. Model Integrity and Malicious Clients: Untrusted or malicious clients can send poisoned model updates to corrupt the entire global model, rendering it useless or planting a backdoor.

3. Performance under Data Heterogeneity (Non-IID): Clients possess data with wildly different distributions (non-IID data), which causes the global model to converge slowly or become heavily biased.

4. Operational Resource Constraints: Communication and computation must be minimized, as clients often have limited bandwidth and processing power

5. Validation in Realistic Settings: Results must be convincing under adverse conditions (e.g., client dropouts, simultaneous attacks).

This prototype provides a practical blueprint for building a trustworthy and deployable FL solution in institutional settings.

1.3 Key Techniques (brief overview)

I considered several families of defenses and optimizations, each with trade-offs:

- **Secure aggregation:** Cryptographic masking or protocols that let the server obtain only the aggregate of client updates, protecting per-client contributions but increasing protocol complexity and overhead.
- **Differential privacy (DP):** Injecting controlled noise at the client or server level to bound information leakage. DP gives formal guarantees but typically degrades accuracy if noise is large.
- **Homomorphic encryption & MPC:** Enable computation on encrypted values to protect updates, offering strong privacy at the cost of heavy computation and higher communication.
- **Robust aggregation:** Aggregators such as trimmed mean or coordinate-wise median limit the effect of outliers and poisoning but may slow learning under benign heterogeneity.

- **Compression and communication reduction:** Quantization, sparsification, and fewer communication rounds lower bandwidth use but can affect convergence and interact with privacy or robustness mechanisms.

Understanding interactions among these techniques will guide architectural and parameter choices for a practical FL system.

1.4 Goals and Objectives

The primary aim is to design and implement a modular, experiment-friendly FL prototype that balances privacy, robustness, and usability. Specific objectives are to:

1. Design a modular separating client, server, aggregation, and privacy modules.
2. Build a Python prototype that simulates clients with non-IID partitions and records metrics for accuracy, convergence, communication cost, and privacy leakage.
3. Implement secure aggregation and configurable DP (local and/or global options).
4. Document trade-offs and produce practical deployment recommendations for institutions like the University of Benin.
5. Publish reproducible code and experiment scripts.

1.5 Scope

I will focus on a research-grade prototype and controlled experiments. Included:

- A server and simulated clients (processes or containers).
- Implementations of secure aggregation and DP.
- Experiments on public benchmark datasets partitioned to emulate non-IID conditions.
- Basic poisoning/Byzantine attack scenarios.

1.6 Methodology

The study will proceed in five phases:

1. Conduct a literature survey to select baseline algorithms and privacy techniques.
2. Produce a system design using object-oriented analysis to define components and interfaces.
3. Implement a Python prototype using standard ML libraries and simulate clients for reproducible experiments.
4. Evaluate across controlled degrees of heterogeneity, client counts, and attack types; measure accuracy, rounds to convergence, communication bytes, and privacy leakage.
5. Analyze results, produce plots and tables, and release code and scripts for replication.

1.7 Challenge

Expected difficulties and planned mitigations:

- **Complex secure protocols:** Begin with lightweight masking schemes before adding heavier cryptography.
- **Accuracy loss from DP:** Use privacy accounting and hyperparameter search to find usable trade-offs.
- **Simulating realistic heterogeneity:** Employ multiple partitioning strategies (label skew, quantity imbalance, feature shifts).
- **Resource limits for experiments:** Use lightweight models for broad parameter sweeps and reserve larger models for focused studies.

1.8 Significance of study

This project will deliver a modular FL prototype and empirical evidence about the trade-offs among privacy, accuracy, and cost. For UNIBEN and similar Nigerian institutions, the work will show practical ways to collaborate on machine learning tasks without sharing raw data. The codebase, experiments, and recommendations will help researchers and decision-makers evaluate when and how FL fits their needs.

1.10 Summary

Federated learning enables multiple parties to jointly train models without exchanging raw data, offering clear benefits for privacy-sensitive domains and distributed research collaborations. To make FL practical for institutions like the University of Benin, we must address non-IID data, privacy leakage, adversarial clients, and constrained resources. This project will design, implement, and evaluate a prototype that clarifies these trade-offs and produces actionable recommendations for real-world adoption.

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction To The Chapter

Federated learning has emerged as a revolutionary approach to address the inherent tension between the need for data-driven insights and the imperative to protect individual privacy. Federated learning (FL) represents a paradigm shift in machine learning, enabling collaborative model training across distributed devices or institutions without centralizing raw data. This approach addresses fundamental privacy concerns inherent in traditional centralized machine learning systems, where sensitive data must be collected, stored, and processed in a single location. Privacy-preserving machine learning (PPML) encompasses a broader set of techniques designed to protect individual privacy while extracting useful insights from data, with federated learning serving as one of its most promising implementations (Yang et al., 2019).

In the African context, and particularly in Nigeria, the adoption of privacy-preserving machine learning technologies represents both a necessity and an opportunity. Nigeria, as Africa's most populous nation and largest economy, generates vast amounts of digital data across multiple sectors, including banking, telecommunications, healthcare, mobile devices and education (Kshetri, 2019). The Nigeria Data Protection Regulation (NDPR), enacted in 2019, established a comprehensive framework for data governance, signaling the country's commitment to international best practices in data protection (Okonkwo, 2020). However, the implementation of advanced privacy-preserving technologies remains nascent, with significant gaps in both technical infrastructure and regulatory enforcement mechanisms. Mobile device manufacturers deploy FL to enhance keyboard prediction, image classification, and personalized recommendations while keeping user data on-device, addressing both privacy concerns and bandwidth constraints (Hard et al., 2018).

Within the Nigerian higher education sector, universities such as the University of Benin face unique challenges in leveraging machine learning technologies while safeguarding student and institutional data. The University of Benin, established in 1970, serves as one of Nigeria's premier institutions of higher learning, with a diverse student population exceeding 40,000 and extensive research activities across multiple disciplines (University of Benin, 2021). The institution's digital transformation initiatives, including learning management systems, student information systems, and research data repositories, generate substantial volumes of sensitive data that require robust protection mechanisms. The implementation of federated learning frameworks in such institutional contexts presents an opportunity to advance data-driven educational innovations while maintaining compliance with data protection regulations and preserving individual privacy rights (Adewumi & Akinyelu, 2021).

This literature review comprehensively examines the theoretical foundations, technological architectures, and practical applications of federated learning for privacy-preserving machine learning. The review traces the evolution of privacy-preserving technologies from traditional anonymization techniques to modern federated learning architectures, analyzes their applicability in the Nigerian as well as global context, and identifies critical gaps in existing research that this study aims to address.

2.1 Federated Learning: Concept and Historical Background

Federated learning is defined as a machine learning approach that trains algorithms across multiple decentralized devices or servers holding local data samples, without exchanging the data itself (McMahan et al., 2017). The core motivation stems from four primary drivers: data decentralization necessitated by privacy regulations, reduction of communication bandwidth requirements, compliance with data sovereignty laws, and mitigation of risks associated with centralized data breaches. Unlike traditional distributed machine learning, which partitions

computation across nodes with access to the same dataset, federated learning operates under the assumption that data remains fundamentally partitioned and cannot be pooled (Kairouz et al., 2021).

The historical trajectory of federated learning builds upon earlier distributed learning paradigms. Early distributed machine learning systems, such as parameter servers and data-parallel training, focused primarily on computational efficiency rather than privacy (Dean et al., 2012). The term "federated learning" emerged in 2016 when Google researchers formalized the approach for training mobile keyboard prediction models across millions of Android devices (McMahan et al., 2017). This seminal work introduced the Federated Averaging (FedAvg) algorithm, demonstrating that models could achieve comparable accuracy to centralized training while keeping data localized. Since 2017, federated learning research has expanded exponentially, with applications extending beyond consumer devices to healthcare consortiums, financial networks, and Internet of Things (IoT) ecosystems (Kairouz et al., 2021; Li et al., 2020).

Federated learning architectures are categorized into three primary types based on data partitioning characteristics. Horizontal federated learning, also termed sample-partitioned FL, applies when datasets across participants share the same feature space but differ in sample space (Yang et al., 2019). For example, hospitals in different cities may hold patient records with identical medical attributes but completely different patient populations. The aggregation process in horizontal FL typically involves weighted averaging of model parameters based on local dataset sizes. Vertical federated learning, or feature-partitioned FL, addresses scenarios where participants hold different features for overlapping sample entities (Yang et al., 2019). A classic example involves a bank and an e-commerce platform sharing customers but possessing complementary data attributes financial history versus purchase behavior enabling collaborative model training through secure entity alignment and feature aggregation. Federated transfer learning combines elements of both paradigms to handle scenarios with differences in both

feature and sample spaces, leveraging transfer learning techniques to extract common representations (Yang et al., 2019). This approach proves particularly valuable in cross-domain applications, such as adapting models trained in data-rich environments to resource-constrained settings.

The architectural distinctions among these FL types necessitate different aggregation strategies, communication protocols, and privacy-preserving mechanisms. Horizontal FL's parameter averaging requires addressing statistical heterogeneity across participants, while vertical FL's feature aggregation demands sophisticated entity resolution and secure intersection computation. Understanding these fundamental paradigms provides the foundation for subsequent discussions of algorithms, privacy mechanisms, and implementation challenges.

In the African context, federated learning adoption remains in early stages but shows promising potential. South Africa and Kenya have led regional initiatives, particularly in mobile health applications and financial inclusion projects (Okolo et al., 2022). Nigeria's journey with privacy-preserving machine learning has been shaped by the country's robust telecommunications sector, growing fintech ecosystem, and increasing digital literacy rates. The Nigerian Communications Commission (NCC) and the National Information Technology Development Agency (NITDA) have emphasized the importance of privacy-preserving technologies in their strategic frameworks, though practical implementation remains limited (NITDA, 2021).

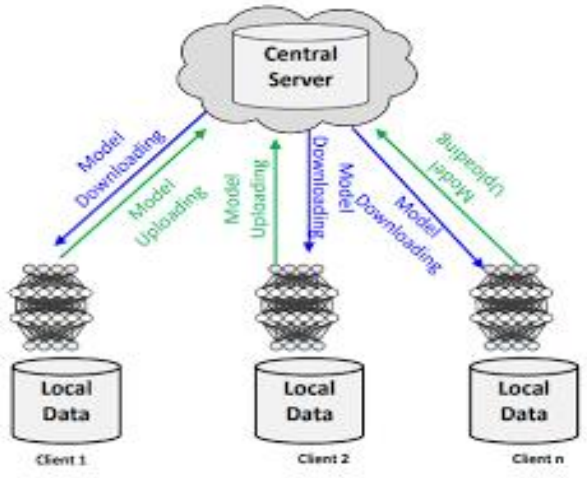

Figure 2.1: Conceptual architecture of federated learning showing distributed clients training on local data and communicating model updates to a central aggregation server.

2.2 Contextualization to Nigerian Educational and Research Setting

The application of federated learning and privacy-preserving machine learning in Nigeria's educational sector presents unique opportunities and challenges shaped by the country's technological landscape, regulatory environment, and institutional capacities. Nigerian universities have increasingly embraced digital transformation initiatives, implementing learning management systems, electronic examination platforms, and data analytics tools for institutional decision-making (Adeoye et al., 2020). However, these developments have occurred against a backdrop of limited cybersecurity infrastructure, inconsistent data governance practices, and gaps in technical expertise (Ochigbo & Osuu, 2021).

The University of Benin exemplifies the challenges and opportunities facing Nigerian higher education institutions in implementing privacy-preserving technologies. The university's data ecosystem encompasses student academic records, research data, health information from the university medical center, financial transactions, and digital learning activities across multiple

platforms. This data holds significant potential for improving educational outcomes through predictive analytics, personalized learning recommendations, early warning systems for at-risk students, and research collaboration across departments. However, the centralized storage and processing of such sensitive information raises substantial privacy and security concerns (Akinyemi & Osuu, 2022).

A federated learning framework implemented at the University of Benin could enable various applications while preserving data privacy. For instance, different departments could collaboratively train predictive models for student performance without sharing individual student records. The university's multiple campuses and affiliated teaching hospitals could jointly develop medical diagnostic models while maintaining patient confidentiality. Research collaborations with other Nigerian universities could proceed without the legal and ethical complications of cross-institutional data transfers (Ezugwu et al., 2021).

The Nigerian Data Protection Regulation (NDPR), implemented by NITDA in 2019, established comprehensive requirements for data processing, including principles of lawfulness, fairness, transparency, purpose limitation, data minimization, accuracy, storage limitation, integrity, and confidentiality (NITDA, 2019). Article 2.1 of the NDPR mandates that data processing must be conducted in a manner that ensures appropriate security against unauthorized or unlawful processing and against accidental loss, destruction, or damage. Federated learning architectures align inherently with these principles by minimizing data movement, processing data locally, and implementing security measures at multiple layers (Okonkwo, 2020).

Figure 2.2: Mapping of NDPR compliance requirements to federated learning implementation components in Nigerian institutional contexts.

Beyond regulatory compliance, the implementation of federated learning in Nigerian universities addresses practical infrastructure challenges. Nigeria's internet connectivity remains characterized by variable bandwidth, intermittent service disruptions, and high data costs, particularly outside major urban centers (Ajao & Oyebode, 2020). Federated learning's ability to operate with asynchronous communication patterns and intermittent client participation makes it particularly suitable for such environments. Clients can perform local training during periods of connectivity and synchronize with the central server opportunistically (Kairouz et al., 2019).

Several Nigerian institutions have initiated exploratory work in privacy-preserving technologies. The University of Lagos established a cybersecurity research center focusing on privacy-enhancing technologies for African contexts (Ajayi et al., 2021). Covenant University has conducted research on secure data sharing mechanisms for healthcare applications in resource-constrained settings (Okoro et al., 2020). The National Centre for Artificial Intelligence and Robotics (NCAIR), established in 2020, has identified privacy-preserving machine learning as a strategic priority area for Nigerian AI development (NCAIR, 2021). However, practical

deployments of federated learning systems in educational or research contexts remain rare, highlighting a significant gap between theoretical interest and operational implementation.

2.3 Traditional Privacy Protection Methods

Before the advent of decentralized learning, privacy protection focused on sanitizing datasets *before* analysis. These methods primarily include techniques aimed at obscuring the identity of individuals within the data:

- **Anonymization and Pseudonymization:** Replacing direct identifiers (like names or IDs) with codes (pseudonymization) or simply removing them (anonymization).
- **Data Aggregation and Generalization:** Combining or grouping data points (e.g., replacing exact ages with age ranges) to make specific individuals harder to distinguish.
- **Data Masking:** Modifying sensitive data fields with random or false values (e.g., replacing credit card numbers with X's).

2.4 Limitations of Traditional Privacy Protection Systems

Traditional methods, while foundational, have proven fundamentally insufficient against modern computational analysis, especially in the context of machine learning:

1. **Susceptibility to Re-identification:** Research has repeatedly demonstrated that even highly anonymized datasets can be **re-identified** by linking them with auxiliary, publicly available information (Narayanan & Shmatikov, 2008). The process of training a powerful predictive model often requires rich, high-dimensional data, making aggressive anonymization which destroys the data's utility unacceptable.

2. **Lack of Process Protection:** Traditional methods protect the static data file, but they do not protect the *process* of learning. Once the model is trained, the model itself can still be interrogated to infer characteristics of the training set, leading to various **model inversion attacks**.

3. **Utility Loss:** To achieve a sufficient level of anonymity against modern attacks, so much information must be removed or distorted that the data becomes unusable for complex tasks, rendering the resulting machine learning model useless (Li et al., 2022).

2.5 Modern Approaches to Privacy-Preserving Machine Learning

The limitations of traditional methods necessitated the development of **cryptographic** and **statistical** defenses that protect data *during* the training process itself. The three dominant modern approaches are:

1. **Homomorphic Encryption (HE):** Allows computation (training) to occur directly on **encrypted data**. The model learns from the encrypted values without ever decrypting them, and the result remains encrypted. While mathematically ideal, HE remains computationally **expensive and slow**, making it impractical for large-scale, deep learning models today.

2. **Secure Multi-Party Computation (SMC):** Allows multiple parties to collectively compute a function (e.g., model aggregation) over their private inputs without revealing any of those inputs to each other. SMC is excellent for small, high-stakes tasks but faces **scalability issues** for coordinating hundreds or thousands of clients, a common scenario in FL.

3. **Federated Learning (FL):** As the most practical and scalable approach, FL relies on statistical decentralization and is often used **in combination** with cryptographic methods (like SecAgg) to maximize security.

2.5.1 Federated Learning

FL's core innovation is achieving privacy by minimizing data exposure. By only sharing model updates, it dramatically reduces the attack surface compared to centralization. The scalability of FL makes it the **dominant practical choice** for edge computing and multi-institutional data collaboration today (Lim et al., 2024).

2.6 Benefits of Modern Privacy-Preserving Approaches

The adoption of modern PPML, particularly FL, delivers strategic advantages that align with the goals of academic and industrial partners:

- **Regulatory Compliance:** FL provides a strong technical measure to meet core data protection principles, such as **data minimization** and **purpose limitation** (NDPA, GDPR).
- **Trust and Collaboration:** By providing a verifiable framework where raw data is never exposed, FL builds trust necessary for competitive or sensitive institutions (e.g., rival banks, different hospital trusts) to engage in valuable joint model development.
- **Reduced Operational Risk:** Eliminating the central data repository removes a massive, single point of failure (SPOF) for malicious attacks, making data breaches less catastrophic.

- **Scalability for Edge AI:** FL is naturally suited for training models on billions of devices (Edge Computing), where centralized data collection is simply infeasible due to network costs and storage requirements (Kairouz et al., 2021).

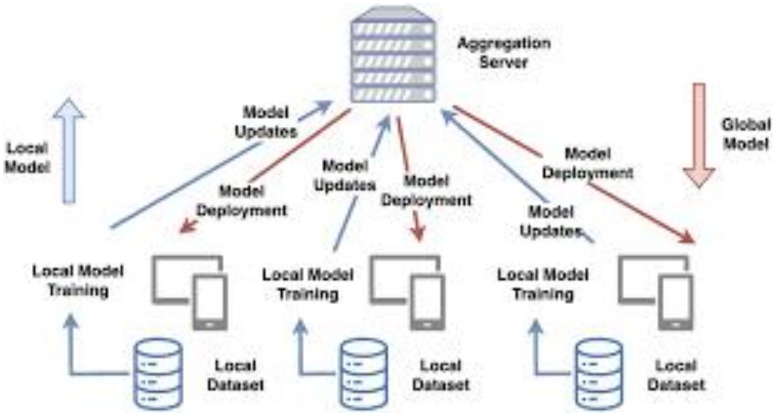

Figure 2.3: Iterative federated learning training process showing multiple rounds of local training, model update communication, and server-side aggregation.

2.7 Privacy Challenges and Regulatory Drivers

Despite its architectural benefits, FL is not inherently a complete privacy solution. Model updates still leak information, making the deployment of *additional* security mechanisms mandatory (Yang et al., 2024).

2.7.1 Key Attack Vectors in FL:

1. **Model Inversion Attacks:** An adversary uses the shared model parameters or updates to reconstruct or infer characteristics of the private training data.
2. **Membership Inference Attacks:** An attacker determines whether a specific individual's data record was included in the training set, violating privacy guarantees.

Regulatory Drivers: The increasing global focus on data rights provides the main impetus for FL research:

- **The Nigeria Data Protection Act (NDPA), 2023:** Mandates the use of appropriate technical and organizational measures (like FL) to ensure the security and confidentiality of personal data.
- **EU General Data Protection Regulation (GDPR):** Its extraterritorial reach affects any Nigerian institution handling data of EU subjects. GDPR requires **Privacy-by-Design**, making FL a necessary default architecture for sensitive projects.
- **Health Insurance Portability and Accountability Act (HIPAA, US):** For medical collaboration, FL is often the only viable way to share knowledge without transferring Protected Health Information (PHI).

2.8 Key Algorithms and Aggregation Methods

The Federated Averaging (FedAvg) algorithm introduced by McMahan et al. (2017) serves as the foundation for most federated learning implementations. FedAvg operates through iterative rounds of local training followed by central aggregation. In each round, the server selects a subset of clients, distributes the current global model, and instructs each client to perform multiple epochs of local training on their private data. Clients compute updated model parameters and transmit only these parameters—not the data—back to the server. The server aggregates client updates through weighted averaging, where weights typically correspond to the size of each client's local dataset, producing a new global model for the next round (McMahan et al., 2017). Mathematically, the global model update is expressed as:

w_{t+1} = Σ(n_k / n) × w_k^{t+1}

where w_{t+1} represents the global model at round t+1, w_k^{t+1} denotes the locally trained model from client k, n_k is the number of samples at client k, and n is the total number of samples across all participating clients.

FedAvg's simplicity and communication efficiency made it widely adopted, but subsequent research identified limitations in handling statistical heterogeneity across clients. FedProx extends FedAvg by adding a proximal term to the local objective function, constraining local updates to remain close to the global model (Li et al., 2020). This proximal regularization, controlled by hyperparameter μ, improves convergence stability when clients possess highly non-IID (non-independently and identically distributed) data:

$$\min F_k(w) + (\mu/2)||w - w^t||^2$$

FedOpt generalizes federated optimization by enabling server-side adaptive optimizers such as FedAdam, FedYogi, and FedAdagrad, which apply momentum and adaptive learning rates to aggregated updates (Reddi et al., 2021). These methods demonstrate superior convergence rates and final model quality compared to FedAvg, particularly in heterogeneous settings, at the cost of maintaining additional server-side state.

Secure aggregation techniques enable privacy-preserving parameter aggregation without revealing individual client contributions. The protocol proposed by Bonawitz et al. (2017) employs secret sharing and cryptographic masking to compute the sum of client updates such that the server learns only the aggregate, not individual values. Clients generate random masks that cancel out in summation, ensuring that even if some clients drop out, the aggregate can still be computed without compromising privacy. This approach provides robustness against honest-but-curious servers and satisfies strong privacy guarantees, though it introduces computational overhead and requires careful handling of client dropouts.

Communication and computation represent critical bottlenecks in federated learning, particularly in cross-device settings with mobile networks and limited bandwidth. Compression strategies address these challenges through various mechanisms. Quantization reduces the precision of transmitted parameters, converting 32-bit floating-point values to lower bit representations (e.g.,

8-bit integers), achieving compression ratios of 4× or higher with minimal accuracy degradation (Alistarh et al., 2017). Sparsification transmits only the most significant parameters or gradients, selecting top-k values by magnitude or applying random masking, with compression ratios exceeding 100× in some scenarios (Lin et al., 2018). Structured updates exploit low-rank matrix factorizations or sketching techniques to reduce communication costs while preserving model expressiveness. These compression methods can be combined with privacy mechanisms, though careful analysis is required to ensure privacy guarantees are maintained under compression.

Decentralized aggregation architectures eliminate the central server bottleneck by enabling peer-to-peer communication among clients. Gossip-based protocols allow clients to exchange and average model parameters with neighbors in a network topology, achieving consensus without centralized coordination (Lalitha et al., 2019). While decentralized approaches offer resilience to single points of failure and potentially reduced communication costs, they introduce challenges in convergence analysis, privacy protection across multiple communication paths, and coordination complexity.

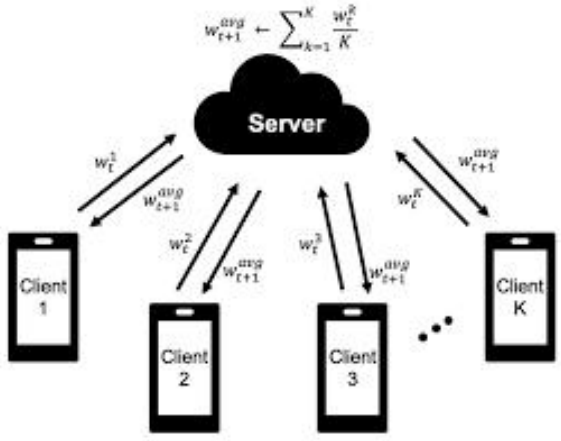

Figure 2.4: Federated Averaging (FedAvg) algorithm flow showing iterative server-client communication pattern, local training, and weighted parameter aggregation.

2.9 Privacy Mechanisms in Federated Learning

FL requires augmentation with specific security protocols to achieve strong privacy guarantees.

Mechanism	Primary Goal	Protection Against	Performance Trade-Off
Differential Privacy (DP)	**Statistical Privacy Guarantee**	Inference and Reconstruction Attacks	Accuracy decreases (utility loss) in exchange for privacy (ϵ).
Secure Aggregation (SecAgg)	**Cryptographic Privacy**	Server or Eavesdropper learning individual updates	Increases communication and computation overhead slightly due to masking protocols.

Differential Privacy (DP):

DP achieves privacy by adding calculated noise to the model updates (client-side) or the aggregated model (server-side). The degree of privacy is quantified by the (ϵ, δ) budget. A lower ϵ means stronger privacy but requires more noise, which degrades model accuracy. Research now focuses on practical DP tools like Opacus (PyTorch) and TensorFlow Privacy (2021-2024), which automatically track the ϵ budget across training rounds.

Secure Aggregation (SecAgg):

SecAgg uses cryptographic techniques (e.g., Shamir's Secret Sharing or Additive Masking) to ensure that the server only receives the sum of all client updates. If a single client drops out, the protocol might fail to decrypt the final sum, a challenge that modern SecAgg protocols have been designed to manage gracefully (Bonawitz et al., 2017; Mo et al., 2021).

2.10 Threats and Robustness

The decentralized nature of FL introduces new vulnerabilities that can be exploited by adversarial clients, requiring strong **robustness** mechanisms.

2.10.1 Byzantine Attacks (Poisoning)

These attacks involve a malicious client attempting to sabotage the learning process. Recent literature categorizes them into two types:

1. **Data Poisoning:** The client taints its local dataset before training (e.g., adding false labels), leading to an update that degrades the model.
2. **Model/Gradient Poisoning:** The client generates and sends intentionally skewed or garbage model updates, often designed to maximize the global model's error (Bagdasaryan et al., 2020).

2.10.2 Robustness Defenses

Defenses against poisoning attacks focus on identifying and mitigating the influence of outlier updates during aggregation:

Defense Method	**Mechanism**	**Strength**	**Weakness**
Krum (Blanchard et al., 2017)	Selects the model update closest to its k nearest neighbors, effectively rejecting distant outliers.	Highly effective against simple poisoning.	Computationally intensive as k increases; struggles with

Defense Method	Mechanism	Strength	Weakness
			clustered attacks.
Trimmed Mean (Yin et al., 2021)	Sorts all model parameters (coordinate-wise) and removes the top and bottom $\alpha\%$ of values before averaging the rest.	Simple, fast, and highly effective against a known percentage of attackers.	Requires an estimate of the number of attackers (α).
Coordinate-wise Median	Replaces the mean aggregation with the median for every single model parameter.	Robust to up to 50% of arbitrary attackers (high theoretical guarantee).	Slower convergence rate than mean-based methods.

A key challenge in modern FL is the **tension between robustness and privacy**. Differential privacy's inherent noise can sometimes mask the signal of a malicious update, making it harder for robust aggregation techniques (like Krum or Trimmed Mean) to filter the poison, requiring careful tuning in the final prototype design.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.0 Chapter Overview

The design of the federated learning (FL) prototype addresses the core challenges while meeting the project goals and scope. The system follows a modular architecture with clear separation of concerns: client, server, aggregation, and privacy components. We focus on enabling secure, privacy-preserving training with realistic non-IID data and limited resources. Key challenges include managing secure aggregation protocols of varying complexity, balancing accuracy with differential privacy, accommodating heterogeneous data across clients, and supporting resource-constrained devices.

The design incorporates contemporary privacy-enhancing technologies including differential privacy (DP) and secure aggregation protocols to address data confidentiality concerns while maintaining model utility. A traceability matrix links project objectives to functional requirements and experimental validation procedures, ensuring systematic coverage of all design goals. The chapter concludes with a statistical analysis plan specifying how experimental results will be rigorously evaluated to demonstrate the system's effectiveness. All design decisions are grounded in recent literature (2020–2025) and informed by qualitative stakeholder insights gathered through preliminary needs assessment.

3.1 System Analysis

3.1.1 Context and Stakeholders

Federated Learning represents a paradigm shift in collaborative machine learning, enabling multiple parties to jointly train models without centralizing raw data. The technology addresses growing concerns about data privacy, regulatory compliance, and the practical limitations of data

centralization in domains such as healthcare, finance, and mobile computing. Recent adoption of FL in production systems including Google's Gboard keyboard prediction and cross-device recommendation systems, demonstrates both the feasibility and commercial value of the approach.

The proposed system targets multiple stakeholder groups with distinct requirements which are

1. **Data owners** (individuals, organizations, or IoT devices) seek to contribute to collaborative learning while retaining full custody of their raw data. These stakeholders prioritize privacy guarantees, minimal computational overhead, and transparency about how their data contributions are used.
2. **Model consumers** (researchers, data scientists, application developers) require trained models with accuracy comparable to centralized baselines, along with verifiable privacy properties and reproducible training procedures.
3. **System administrators** need deployment flexibility, monitoring capabilities, and security controls suitable for enterprise or institutional environments.
4. **Regulatory and ethics bodies** demand compliance with data protection frameworks (GDPR, HIPAA) and mechanisms for informed consent and audit trails.

A preliminary needs assessment was conducted using an online survey instrument to inform system design priorities and constraints. The instrument(survey) yielded **N = 11** responses collected between **31 October 2025 and 1 November 2025**. Respondent roles were dominated by data subjects and end-users: **7/11 (63.6%)** self-identified as End-users / Data Subjects, **3/11 (27.3%)** as Researchers/Developers, and **1/11 (9.1%)** as IT/System Administrator or Data Owner. Organization types (N = 10 non-missing) included universities/research laboratories (**4/10; 40.0%**) and small or medium businesses (**3/10; 30.0%**). Device heterogeneity was evident: respondents reported availability of mobile phones for local training (**3/11; 27.3%**),

small ARM/Raspberry-Pi class devices (**2/11; 18.2%**), desktop/server CPUs (**2/11; 18.2%**), and GPU servers (**1/11; 9.1%**); one respondent indicated no local compute capability (**1/11; 9.1%**). Reported data modalities were dominated by text/documents (**5/11; 45.5%**), with limited reporting of sensor/IoT logs. Preferences regarding data locality were strong: among respondents who answered (N = 10), **8/10 (80.0%)** preferred that raw data remain on-device (5/10 conditional on protections such as encryption/noise; 3/10 unqualified), while **2/10 (20.0%)** were unsure. Technical readiness for differential privacy (DP) and accounting was limited: among respondents who answered (N = 6), responses clustered at neutral/partial readiness (3/6 rated neutral), with one respondent each indicating lower or higher readiness. Overall, these quantified findings motivated design choices prioritizing support for resource-constrained clients, configurable privacy utility trade-offs, and communication protocols tolerant of intermittent client participation.

3.1.2 Survey methods & limitations

The survey collected basic demographic and infrastructural information and used a mixture of closed and open questions; timestamps indicate a data-collection window from **2025-10-31** to **2025-11-1**. Recruitment metadata (channels, sampling frame, and response solicitation method) is not recorded in the data collected; therefore the sample should be treated as convenience / self-selected. Key limitations are:

- the small sample size (N = 11),
- incomplete responses for multiple questions (several items have substantial missingness),
- potential bias toward academic and small-organization perspectives.

Consequently, the survey results are **indicative** rather than statistically representative; design decisions driven by these results are expressed in Chapter 3 as targets to be validated empirically in the experimental plan.

3.1.3 Analysis of Existing Systems

Traditional centralized machine learning requires aggregating training data in a single location, creating several critical vulnerabilities. **Privacy risks** arise when sensitive data must be transmitted to and stored in centralized repositories, exposing it to potential breaches, insider threats, and regulatory non-compliance. Even with encryption in transit and at rest, the centralized model inherently increases attack surface and contradicts principles of data minimization. **Scalability bottlenecks** emerge as data volumes grow; transferring petabytes of raw data from distributed sources to central servers becomes prohibitively expensive in terms of bandwidth, latency, and cost. **Regulatory barriers** increasingly prevent data centralization, particularly in healthcare (HIPAA) and financial services (GDPR, CCPA), where cross-border data transfers face strict limitations.

Early distributed learning approaches, such as parameter servers and data-parallel training, reduce some scalability concerns but fail to address privacy fundamentally. These systems still require data owners to trust a central coordinator with either raw data or high-fidelity gradients that can leak information through gradient inversion attacks. Recent studies demonstrate that even aggregated gradients can reveal training data properties, including membership inference, attribute inference, and in some cases reconstruction of individual training examples.

Existing FL frameworks including TensorFlow Federated (TFF), PySyft, and Flower that provide foundational infrastructure but exhibit limitations for production deployment. TFF offers robust simulation capabilities and tight integration with TensorFlow but lacks built-in secure aggregation for production use and requires significant expertise to deploy beyond single-machine simulations. PySyft pioneered privacy-preserving FL with cryptographic protocols but has faced challenges with API stability, documentation, and scalability to large client populations. Flower provides a lightweight, framework-agnostic FL server but delegates privacy and security mechanisms to users, requiring custom integration of DP and secure aggregation. None of these

frameworks provide integrated, production-ready solutions combining differential privacy, secure aggregation, and robust aggregation in a single cohesive system with institutional deployment tooling.

3.1.4 System Challenges and Requirements

Several key design challenges drive the system requirements:

- Secure aggregation complexity: Implementing secure aggregation can range from simple additive masking to heavier cryptographic schemes. We begin with a basic pairwise masking protocol: each client adds random masks to its model update, which cancel out when aggregated, so the server learns only the sum of updates. This ensures no individual update is revealed. The modular design allows substituting stronger cryptography (e.g. homomorphic encryption or multi-party computation) if needed.

- Privacy-utility trade-off (Differential Privacy): Differential privacy (DP) protects against inference on model updates but introduces noise that can degrade accuracy. Each client clips its gradient and adds calibrated Gaussian noise to its update. A privacy accountant (e.g. moments accountant with Rényi DP) composes the per-round privacy loss to compute the total (ε,δ) budget. By tuning the clipping norm and noise scale, we balance privacy versus accuracy according to a target budget.

- Data heterogeneity: In real FL, client data are non-IID: label distributions, feature domains, and sample sizes vary across clients. Such heterogeneity is known to slow convergence and reduce accuracy. We explicitly simulate non-IID conditions by partitioning datasets so that each client has uneven class distributions or feature shifts. This allows us to measure how convergence and final accuracy are affected by label skew, feature shift, and data imbalance during training.

- Limited computation and communication: Many clients (e.g. mobile or IoT devices) have constrained memory, computation, and bandwidth. The design therefore uses lightweight model architectures for initial parameter sweeps and larger models for final evaluation. Communication efficiency is addressed via update compression (e.g. top-k sparsification or quantization). We aim for per-round payloads on the order of a few megabytes (for models with millions of parameters), since a full float32 update could otherwise be tens of MB. These constraints influence our choices in model size and protocol.

These challenges map to functional requirements: the system must support FedAvg (and variants like FedProx), client-level DP, secure aggregation (additive masking), and robust aggregation rules (e.g. trimmed-mean or Krum). Non-functional requirements include achieving test accuracy close to centralized training (within a few percentage points), convergence in a reasonable number of rounds, and maintainable communication overhead. We capture these in a requirements traceability matrix (omitted here for brevity).

3.2 Requirements Analysis

3.2.1 Functional requirements

Table 3.1 lists the system's functional requirements derived from stakeholder needs and problem analysis. Each requirement has a priority (Critical / High / Medium) and the primary system component responsible for implementation.

Table 3.1 — Functional requirements

Req ID	Requirement (simple)	Priority	Component

FR1	Implement FedAvg and FedProx for model aggregation.	Critical	Aggregation Module
FR2	Apply client-level differential privacy (configurable ε, δ).	Critical	Privacy Module
FR3	Provide secure aggregation (additive masking) for client updates.	High	Security Module
FR4	Support gradient compression (top-k sparsification).	High	Communication Module
FR5	Provide robust aggregation (trimmed mean, Krum) to mitigate Byzantine updates.	High	Aggregation Module
FR6	Support multiple data modalities (tabular, text, images).	Medium	Data Handler
FR7	Expose REST API for client registration, model distribution and control.	Critical	Server API
FR8	Log training rounds and events with configurable verbosity.	Medium	Logging Module
FR9	Support asynchronous client updates with staleness handling.	Medium	Scheduling Module
FR10	Produce convergence plots and privacy-accounting reports automatically.	High	Evaluation Module

3.2.2 Traceability matrix

Table 3.2 maps project objectives to the functional/non-functional requirements they support and to the experiments that validate each mapping.

Table 3.2 — Traceability matrix

Objective	Related requirements	Validating experiments
Obj-1: Privacy-preserving FL	FR1, FR2, FR3; NFR1, NFR2	EXP-1 (DP sweep), EXP-2 (utility vs privacy), EXP-4 (baseline)
Obj-2: Accuracy close to centralized	FR1, FR5; NFR3	EXP-2, EXP-4, EXP-5 (non-IID)
Obj-3: Communication efficiency	FR4; NFR4	EXP-3 (compression), EXP-6 (participation scenarios)
Obj-4: Security vs malicious clients	FR3, FR5; NFR5	EXP-7 (Byzantine injection), threat-model validation
Obj-5: Heterogeneous clients	FR6, FR9; NFR6	EXP-6 (device profiling and participation)
Obj-6: Production readiness & auditability	FR7, FR8, FR10; NFR7	Integration tests, audit checklist, usability evaluation (Section 3.11)

3.2.3 Non-Functional Requirements

Non-functional requirements (NFRs) define measurable quality attributes, operational constraints, and acceptance criteria for the federated learning (FL) system. Each NFR below states a target and the empirical procedure that will determine the final numeric value.

NFR1: Privacy guarantee (differential privacy).

The system targets configurable (ε, δ)-differential privacy on client updates. A parameter sweep will evaluate $\varepsilon \in \{1.0, 3.0, 10.0\}$with $\delta = 1 \times 10^{-5}$; $\varepsilon = 3.0$is the working target pending empirical validation. Privacy accounting will use moments-accountant / Rényi DP analysis

(Opacus implementation), explicitly recording sampling probability, clipping norm, and composition across rounds. Final parameters will be selected from EXP-2 results and bounded by any applicable institutional policy.

NFR2: Model utility.

The global model should achieve test accuracy within 5 percentage points of an equivalent centralized baseline on the same held-out test set. This acceptance target will be validated in EXP-2 and EXP-4, which quantify utility loss across ε, non-IID severity, and client participation. Reported utility metrics must include mean and standard deviation across independent seeds.

NFR3: Convergence rounds.

Convergence is targeted within Rcommunication rounds, where $R \in \{100,300\}$depending on dataset complexity and non-IID degree; $R = 200$is the working value for moderate non-IID. Round counts will be tuned in EXP-5 to balance convergence, wall-clock time, and client compute.

NFR4: Communication efficiency.

Per-round upstream transmission per client is targeted to remain ≤ 10MB for models up to 10million parameters. This operational estimate assumes 32-bit floats (4 bytes/parameter) and top-k sparsification with $k = 10\%$(≈4 MB pre-compression for 10M parameters). EXP-3 will measure actual bytes transmitted including protocol overhead, signatures/metadata, and retransmissions; empirical compression ratios will be recorded.

NFR5: Byzantine resilience.

Robust aggregation methods will be evaluated with the objective of tolerating up to 20% malicious clients under the stated threat model (see section 3.4). Candidate methods include trimmed mean, coordinate-wise median, and Krum; final selection will trade off robustness and computational cost and will be validated by adversarial injection tests in EXP-7. The stated

tolerance is conditional on the threat assumptions (e.g., adversary independence, participation patterns); failure modes are discussed in section 3.4.

NFR6: Client resource constraints.

Client software must operate under constrained resources by defaulting to a conservative per-process memory budget of $\leq$ 2GB and a single CPU core for local training. The client will expose lightweight profiles (e.g., micro-client for inference/partial aggregation, standard client for training) and graceful fallbacks (reduced batch size, fewer local epochs, server assistance). Pilot profiling (Section 3.11) will validate resource budgets and profile efficacy.

NFR7: Auditability and compliance.

Operational telemetry and audit logs must record client registration, round participation, aggregation events, and privacy accounting metadata with timestamps and pseudonymized identifiers. Logs shall be exportable in structured JSON and support retention/erasure policies consistent with applicable regulations. Logging must avoid retention of raw data or information that materially increases re-identification risk; access to linkages shall be controlled by policy.

3.3 Use cases and user stories

This section lists core use cases that capture primary federated-learning interactions. Each entry states actors, preconditions, a concise main flow, and the expected outcome.

Use Case ID	Title	Actors	Preconditions	Main flow (condensed)	Outcome
UC1	Client registration	Data owner → Server	Client has network access	Client registers (sends public key); server validates and issues client ID and model metadata.	Client enrolled and able to participate.
UC2	Local training round	Client ⇄ Server	Client registered; global model available	Client downloads model, performs E local epochs, clips gradients, applies DP noise, uploads masked update.	Masked client update prepared and transmitted.
UC3	Secure aggregation	Server, multiple clients	≥ t masked updates received	Server executes unmasking protocol and computes weighted FedAvg.	New global model computed without observing individual updates.
UC4	Model evaluation	Model consumer	Global model snapshot available	Consumer retrieves snapshot and evaluates on holdout test set;	Model accuracy and privacy budget reported.

				privacy accounting attached.	
UC5	Byzantine mitigation	Server, adversarial client(s)	Corrupted or outlier updates present	Server applies robust aggregation (Krum / trimmed mean), flags extreme updates.	Poisoning impact reduced; model integrity preserved.
UC6	Privacy-budget monitoring	System administrator	System operational	Admin queries accounting module; module reports cumulative ε and per-round logs.	Privacy budget tracked; compliance actions possible.

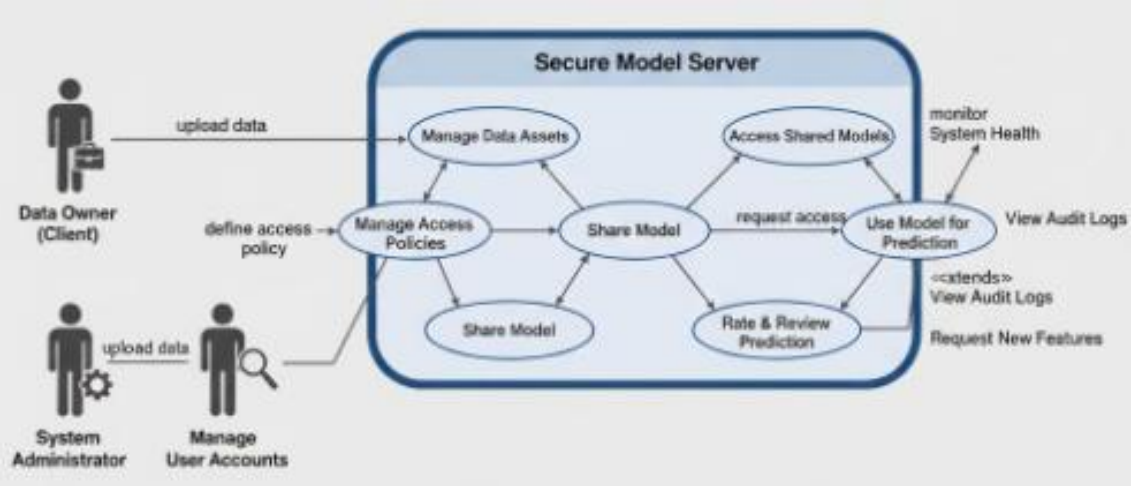

Figure 3.1: UML use-case diagram illustrating interactions among Data Owners (clients), Server, Model Consumers, and System Administrators.

3.4 Proposed system overview and architecture

The FL system is built on a client-server model with modular components. Each **Client** holds private data and a local model. In each training round, the server selects a subset of available clients, sends them the current global model, and awaits their updates. Each client computes local training (e.g. mini-batch SGD on its data) and produces a model update (gradients or parameter delta). Before sending, the client optionally applies gradient compression (e.g. sending only the top-k largest updates) to reduce bandwidth, and then adds DP noise or prepares masks as configured.

Figure 3.2: High-level architecture showing five primary components

The FL **Server** orchestrates training rounds. It maintains the global model, selects clients, and collects updates. The server tolerates client dropout: if some clients do not respond, it proceeds with the received updates. Once a round's updates are in, the server invokes the **Aggregation Module**, which by default performs Federated Averaging (FedAvg) – a weighted mean of the clients' updates. We also support FedProx (adding a proximal term) to stabilize training under heterogeneity. Robust aggregation (median or trimmed mean) can be enabled if poisoning is detected.

The **Privacy Module** enforces differential privacy. It clips each client's update to a fixed L2 norm and adds Gaussian noise to each parameter before aggregation, ensuring that the overall

training satisfies a target (ε,δ)-DP guarantee. A privacy accountant (using, e.g., Rényi DP) tracks the cumulative privacy loss across rounds. By encapsulating DP logic in this module, we can easily adjust noise and clipping parameters for different experiments.

The **Security Module** implements secure aggregation. Clients exchange seed keys and mask their updates before sending. The server receives only masked updates and cannot inspect them individually. When the server sums the masked updates, the random masks cancel out, revealing only the sum of the true updates. This prevents a curious server from learning any client's data beyond the DP guarantee. The protocol is designed to tolerate dropouts: if some clients drop out, remaining clients adjust their mask contributions so the cancellation still works.

Logging and APIs provide system visibility. Training rounds, model versions, and privacy metrics (ε,δ) are logged for audit. Clients interact via a REST API for registration, model download, and update upload. This infrastructure supports monitoring (e.g. tracking client participation) and ensures reproducibility: experiment configurations and logs can be saved and inspected.

By separating concerns into modules (Client, Server, Privacy, Security, Aggregation), the prototype remains flexible and testable. For example, one can run the aggregation module on fixed inputs without simulating clients, or swap the privacy module with a dummy (no noise) to isolate the effect of DP.

3.4.1 Deployment topologies

The system supports three practical deployment styles to meet institutional and network needs:

- **Cloud:** Server and services run on cloud VMs or managed services. Clients connect over the public Internet (HTTPS/TLS). Cloud hosting allows elastic scaling of orchestration and storage.

- **On-premise:** Server and services run inside an institution's network for strict data-governance or regulatory needs. Integrates with internal auth (LDAP/AD) and private networking.
- **Hybrid:** A cloud coordinator handles a mix of internal and external clients. Firewalled clients use VPNs or HTTP(S) proxies. Pilot runs should use isolated sandboxes before full roll-out.

Common operational attributes across all topologies:

- Transport always uses TLS with modern cipher suites; mutual TLS can be enabled where required.
- Server components are containerized (Docker) and deployed via orchestration (Kubernetes / Docker Compose) for reliability and scaling.
- Clients are provided as lightweight installers or binaries; a reduced-footprint client profile supports constrained devices.
- Deployments are configured declaratively (service endpoints, privacy/aggregation settings, logging) for repeatability and audits.
- Rollouts follow staged pilots, health checks, retry/time-out policies and integrated monitoring for observability.

3.5 Data Flow And Per-Round Sequence

A federated training round follows these steps:

1. **Select clients.** Server samples an available subset according to policy and availability.
2. **Send model.** Server distributes the current global model (or delta) to selected clients.

3. **Local training.** Each client runs local epochs on private data and computes an update (weight delta).
4. **Clip & DP.** Clients clip the update norm and (if enabled) add calibrated DP noise per the configured budget.
5. **Masking (optional).** Clients apply additive masks for secure aggregation and upload masked updates.
6. **Collect & unmask.** Server gathers updates and, if the unmasking threshold is met, removes masks to obtain the aggregate.
7. **Aggregate.** Server computes the global update (FedAvg or a robust estimator) and updates the global model.
8. **Account & log.** Server advances privacy accounting, logs per-round metadata (pseudonymized IDs, timestamps, metrics) and exports monitoring metrics.
9. **Finish.** Server increments the round and schedules the next cycle.

Key operational rules and resilience measures:

- **Data locality:** Raw data never leaves clients; only clipped/noisy/masked updates are transmitted.
- **Fault tolerance:** Timeouts, retries, partial-participation handling and staleness policies enable progress when clients drop out or are slow.
- **Adversary resilience:** Robust aggregation, outlier detection and masked aggregation limit poisoning and data leakage; unmasking thresholds and failure events are monitored and logged.

This linear flow is the canonical round; implementations may adopt asynchronous updates, hierarchical aggregation, or batching to better fit scale and network constraints.

3.6 Federated Learning Algorithms and Modules

3.6.1 Federated Averaging (FedAvg)

In each FL round, the server sends the current global model to a selected set of clients. Each client trains this model locally for a few epochs on its private data, then computes a weight update (the difference between the new and old model) and sends it back. The server aggregates these updates by a weighted average (weights proportional to each client's data size) to form the new global model. Conceptually, this is *iterative model averaging*: no raw data leaves clients, only model updates. This process repeats until convergence. (This procedure follows the classic FedAvg approach.)

- *Local training:* Clients perform standard SGD on their private data.
- *Aggregation:* Server computes $\displaystyle w^t = w^{t-1} + \sum_{k\in C_t} \alpha_k,\Delta w_k^t$, where α_k is the client's data-fraction weight.
- *Costs:* Each round involves local compute proportional to $O(E\cdot|D_k|\cdot M)$ (epochs E, data size, model size M), and communication of M parameters (download) and M parameters (upload), i.e. about $2M$ floats per round.

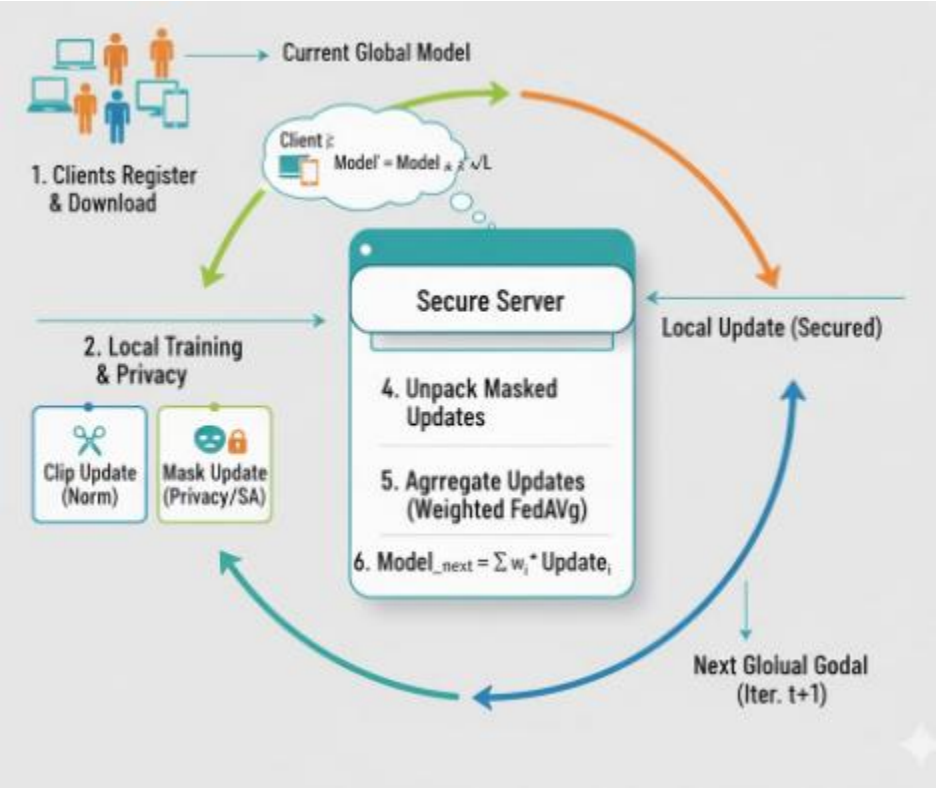

Figure 3.3: Federated Learning Workflow: Clients register, download the global model, and perform local training. They then clip and mask updates for privacy before uploading. The server unpacks, aggregates (e.g., via weighted FedAvg), and generates the next global model.

Handling heterogeneity (FedProx): If client data are highly non-IID, standard FedAvg may converge slowly. An extension called *FedProx* adds a proximal term to each client's loss, penalizing deviation from the global model. In practice, this means each client's local loss includes $\tfrac{\mu}{2}|w - w^{t-1}|^2$, which encourages updates to stay close to the global model. This stabilizes training under heterogeneous data; it can often reduce the number of rounds needed to reach the same accuracy.

Client-Side Differential Privacy: To limit what any client's data reveal, **differential privacy (DP)** can be applied to the model updates. Each client clips its update to a fixed L_2 norm and adds random noise before sending it. In practice, a client computes its raw update Δw_k,

then sets $\tilde{\Delta w}_k = \Delta w_k \cdot \min(1, C/|\Delta w_k|_2)$
for a clipping norm C. This bounds the sensitivity of any single client. The client then adds Gaussian noise: $\Delta w'_k = \tilde{\Delta w}_k + \mathcal{N}(0,,\sigma^2 I)$,
where the noise scale σ is chosen so that the sequence of rounds satisfies (ϵ,δ)-DP for some target ϵ (privacy budget). A privacy accountant (e.g. Rényi or moment accounting) tracks the cumulative ϵ over rounds based on the sampling rate, C, and σ.

DP summary: Clip each local update to norm C and add calibrated Gaussian noise. This guarantees that the presence or absence of any individual's data has only a bounded effect on the aggregated model. Tuning C trades off utility versus privacy: too small C truncates useful updates, too large C requires more noise. Tools like Opacus (PyTorch) or TensorFlow Privacy can compute the required σ to achieve a given (ϵ,δ). In short, *Gaussian noise addition is a standard way to achieve differential privacy*.

3.6.2 Secure Aggregation

Secure aggregation ensures the server learns only the **sum** of client updates, not individual contributions. We use an **additive masking** protocol: each client k establishes pairwise secrets with other clients (e.g. via Diffie–Hellman) and derives mask vectors $s_{k,j}$ with each other client j. Client k computes a combined mask $M_k = \sum_{j\neq k} s_{k,j}$ (taking half with a plus sign and half with minus to cancel later). It then sends $u_k = \Delta w'*k + M_k$ to the server. When the server sums all u_k, the pairwise masks cancel out (as $s*{k,j} + s_{j,k}=0$), leaving only $\sum_k \Delta w'_k$ – the noisy sum of updates. Individual $\Delta w'_k$ remain hidden.

This protocol assumes at least a threshold t of honest clients complete the round so that masks pair off correctly. Clients must not collude with the server (or the protocol must assume honest-

but-curious server and at least two honest clients). In practice, dropout-resilient variants (using secret-sharing to allow unmasking even if some clients drop out) are used. The cost is moderate: clients do $O(|C_t|)$ pairwise key exchanges (amortized) and mask additions of size M, and communication remains sending M masked parameters (no extra bandwidth). The benefit is that the server can only decrypt the aggregate, preserving privacy of individual updates.

3.6.3 Robust Aggregation

To defend against malicious (Byzantine) clients that send poisoned updates, we use *robust aggregation*. Simple averaging is vulnerable if some clients send extreme values. Alternative rules include:

- **Coordinate-wise trimmed mean:** For each model parameter (coordinate), sort the client values, remove the largest and smallest fraction (say $\beta=20%$) and average the rest. This discards outliers. In ideal conditions it can tolerate up to $\beta<50%$ of outliers. Its cost is $O(|C_t|\log|C_t|)$ per coordinate, but implementation can be optimized.
- **Coordinate-wise median:** Take the median of client values at each coordinate. This is highly robust (requires at least half honest), but computing medians is slightly more expensive ($O(|C_t|)$ per coordinate, but with a hidden constant).
- **Krum:** This selects one client's update that is most "central" to others (minimizing sum of distances to nearest neighbors). It requires knowing or estimating the number f of Byzantines. Krum is robust but has $O(|C_t|^2M)$ cost (pairwise distances), so it's used only if $|C_t|$ is small.

In practice, we plan to use trimmed-mean by default (e.g. trim 20%) because it gives good robustness with moderate cost. These robust rules limit the influence of poisoned updates so the global model remains accurate despite malicious clients.

3.6.4 Communication Efficiency

To reduce uplink/downlink overhead, we apply **model compression**:

- **Top-k sparsification:** Each client sends only the largest-magnitude ρM parameters of its update (plus their indices), setting others to zero. For example, with $\rho=0.1$, only 10% of parameters (say 1 M out of 10 M) are sent. This cuts upload from ~80 MB down to a few MB (e.g. 1 M parameters × 4 bytes = ~4 MB).
- **Quantization:** We also compress parameter precision. For instance, use 8-bit or 16-bit floats instead of 32-bit. After top-k, quantizing values further reduces size. In our example, 1,000,000 parameters at 8 bits → ~1 MB (plus index overhead).

Combining sparsification and quantization yields order-of-magnitude savings (10–40× smaller). The compute cost on the client is modest: selecting top-k takes $O(M\log k)$ or uses approximate algorithms, quantization is $O(M)$. There is little accuracy impact for moderate compression levels (existing studies report <1–2% loss for 10% sparsity with quantization). We will measure actual bytes per round (including protocol headers) in experiment EXP-3 and tune (ρ, bits) to meet our bandwidth NFRs.

3.7 System Architecture and Components

The system is modular and scalable, following a client-server FL design. Key components (shown in a component diagram) include:

- **FL Server (Orchestrator):** Manages global model state, client selection, scheduling rounds, and invoking the aggregation engine. It stores model versions, tracks privacy budget, and provides admin APIs.
- **Client Trainer:** Runs on each data-owning device. It pulls the latest global model, runs local training (for E epochs), clips its update, injects DP noise, optionally applies

secure masking, and submits the final update. We plan "lightweight" client modes (e.g. reduced batch size, sparse training) for resource-constrained devices.

- **Privacy Module:** Handles DP-specific tasks on the server side: calculating noise scale, tracking cumulative privacy loss (using RDP/moments accountant), and logging privacy metadata. It centralizes accounting to prevent accidental DP errors.
- **Security Module:** Implements cryptographic protocols for secure aggregation. It manages key exchanges, mask generation and unmasking, and integrity checks. This module is pluggable: one can swap in threshold-cryptography or trusted-execution alternatives if needed.
- **Communication Layer:** Manages serialization and transport. It applies compression (top-k, quantization) before sending, and handles retries, timeouts, and network issues (TLS encryption, proxy support). It exposes gRPC/REST APIs for clients.
- **Evaluation Module:** Collects metrics (accuracy, loss, privacy spent), generates plots (convergence curves, budget timelines), and exports reports for audits.

Each component has clear responsibilities and interfaces. For example, the FL Server calls aggregate_updates(), which may invoke FedAvg or a robust aggregator; PrivacyAccountant tracks ϵ. Containers (Docker) and CI pipelines will be used to deploy these components. *Figure: Component Diagram illustrating FL Server, Client Trainer, Privacy & Security modules, Communication layer, etc.* (A UML component diagram here would visually show these modules and their interfaces.)

3.8 Threat Model and Security

We assume an **honest-but-curious server** and potentially malicious clients, with these core assumptions:

- **A1: Honest-but-curious server.** The server follows the protocol but tries to learn data. Secure aggregation and client-side DP protect client updates under this assumption.
- **A2: Minimum honest clients per round.** We assume at least $t\ge2$ honest participants complete each round. This ensures pairwise masks can cancel in secure aggregation. (We plan dropout-resilient masking protocols so long as dropouts are below threshold.)
- **A3: No collusion.** We assume clients do not collude with each other or the server. If collusion is possible, stronger crypto (like homomorphic encryption or TEEs) would be needed.
- **A4: Bounded Byzantine fraction.** Only a limited fraction of clients may be malicious per round. Robust aggregation (e.g. trimmed mean) will tolerate a certain fraction (e.g. up to ~20%) of outliers without harming the model.

Under these assumptions, we achieve *defense-in-depth*:

- **Confidentiality:** Secure aggregation masks individual updates, so the server sees only $\sum \Delta w'_k$. Combined with DP, even the aggregate leaks limited information (bounded by ϵ).
- **Integrity:** Robust aggregation ensures that a few adversarial updates cannot significantly skew the model. Outliers are trimmed or medians taken coordinate-wise.
- **Availability:** The protocol tolerates client dropouts (via resilient secure-aggregation) and asynchronous rounds, so training can continue even if some clients are offline.

We explicitly map adversary capabilities to mitigations (see Table). For example, if a semi-honest server tries to inspect updates, secure masking prevents it; if many clients drop out, robust unmasking or partial aggregation strategies keep progress.

3.9 Testing and Validation Plan

We will empirically validate the system with a suite of experiments covering functionality, performance, and privacy-utility trade-offs. Key elements include:

- **Datasets:** Standard benchmarks such as MNIST, CIFAR-10, and FEMNIST (Federated EMNIST) will be used. These cover image tasks and have natural/non-IID client partitions. A synthetic tabular dataset will allow quick, controlled experiments.
- **Partitioning:** We will test both *IID* (data randomly split) and various *non-IID* scenarios. For example, label-skew (clients have samples from only some classes) and feature-skew (client data distributions differ). FEMNIST provides a real-world non-IID partition by writer.
- **Baselines:** We compare against centralized training (pool all data) as an upper-bound, and standard FedAvg without any privacy or robustness as a baseline. This isolates the impact of our enhancements.

Experiment examples:

- *Privacy sweep (EXP-1):* Vary DP ϵ (e.g. 1.0 to 10.0) and clipping norm to measure the effect on accuracy and cumulative privacy loss.
- *Privacy-utility (EXP-2):* Compare FedAvg vs. FedAvg+DP on accuracy for MNIST and CIFAR-10, to see how much accuracy drop DP incurs.
- *Compression study (EXP-3):* Measure bytes transmitted per round under no compression, top-5%, 10% sparsity, 8/16-bit quantization. Evaluate impact on convergence.
- *Non-IID robustness (EXP-5):* Compare FedAvg vs. FedProx under different label-skew severities, and measure convergence gap.

- *Dropout dynamics (EXP-6):* Vary participation fraction (e.g. 30%, 50%, 90%) and measure rounds-to-converge.
- *Byzantine resilience (EXP-7):* Simulate malicious clients (e.g. random noise or flipped gradients) at different fractions; compare FedAvg vs. trimmed-mean vs. Krum in final accuracy.

Each experiment runs multiple seeds (≥5) for statistical confidence. We will report means and 95% confidence intervals (via bootstrap) for accuracy, privacy ϵ, and communication cost per round. Standard hypothesis tests (paired t-test or Wilcoxon) will assess significance of differences between methods. Results will be tabulated and plotted (with shaded CI regions) to transparently show trade-offs. For example, *"FedAvg+DP (ε=3.0) achieved 94.2%±0.8% accuracy (95% CI) after 100 rounds"*, with comparisons against baselines.

3.10 Implementation and Deployment

For implementation, we follow best practices to ensure reproducibility and security:

- **Secure aggregation:** We will implement an additive-masking protocol as the baseline. Clients exchange keys (e.g. Diffie–Hellman) to derive shared masks; each client masks its update before upload. This requires no trust in the server seeing individual updates. We will use existing crypto libraries (e.g. cryptography.hazmat) for key generation and a secure PRG for masks. The protocol will be robust to moderate dropouts (e.g. using the Bonawitz et al. method). We will document all assumptions (honest-but-curious server, honest participants). As an alternative, we may experiment with homomorphic encryption (HE) libraries like TenSEAL, noting the significant overheads. A high-level pseudocode of the masking protocol will be included in the appendix.

- **Simulation vs real deployment:** For development, we will use a simulation framework (e.g. Flower or custom Python threads) to mimic many clients on one machine. For real-world testing, we will run clients in isolated containers/VMs. We will script containerized deployments: server components in Docker (or Kubernetes for scaling), and client software as a pip-installable package or lightweight executable.

- **CI/CD and reproducibility:** We will pin all library versions (Python 3.9+, PyTorch/TensorFlow 2.x, etc.) and track them in requirements.txt. Experiment configurations (hyperparameters, random seed, code version) will be logged so results are reproducible. Every commit triggers CI that runs unit tests and integration smoke tests. Successful builds will publish Docker images and record hashes for traceability.

- **Quality checks:** Automated checks will enforce code style (linters), run unit tests for critical modules (FedAvg, DP injector, secure agg), and integration tests simulating a small FL round. We will also optionally run a mini-experiment in CI to catch obvious bugs early.

- **Alternatives & constraints:** We will maintain flexibility to swap components: aggregation method (FedAvg, FedProx, robust), privacy accountant (Opacus vs TF-Privacy), crypto backend, etc. Clients will have "lite" profiles (smaller models, batch sizes) for low-power devices. For restrictive networks, we will use HTTPS with proxy-friendly settings; a fallback agent (e.g. polling-based) may be provided for extreme cases. All DP configurations and crypto parameters will be reviewable via admin dashboards.

3.11 System Testing and Usability

We will perform multi-level testing:

- **Unit Tests:** Each core algorithmic function (FedAvg aggregator, DP noise injector, secure mask routines) has unit tests. Cases include no-clients, one-client, all clients drop out, extreme DP values. We use property-based tests (Hypothesis) to stress edge cases. Aim: 100% pass, high code coverage.
- **Integration Tests:** Simulate client-server interaction (mock clients API). Check correct API responses, that model updates flow through correctly, and privacy/accounting data is recorded. We will test partial participation, timeouts, malformed messages.
- **System Tests:** Run end-to-end FL training on a small dataset (e.g. 1000 MNIST samples, 10 clients) to ensure convergence. Smoke tests in CI ensure no crash. We verify final model reaches expected accuracy and privacy budget is reported.
- **Performance Tests:** Using tools like Locust or JMeter, simulate up to 500 concurrent clients pushing updates. Measure server throughput and latency (p50, p95, p99). We set targets (e.g. p95 < 5s, sustain ≥50 updates/sec on an 8-core CPU). We profile and optimize if needed (e.g. async aggregation, load balancing).
- **Security Tests:** Attempt known attacks: e.g. gradient inversion on updates (should fail or be weak under DP), membership inference (should have accuracy near random at planned ε), Byzantine poisoning (see EXP-7), and unauthorized access (using invalid tokens) – these should be detected or blocked. For each, we check logs and alerts.
- **Usability Evaluation:** We will gather feedback from a small group of users (developers and admins). They will install a client, run a training session, and use the admin dashboard. We will apply the System Usability Scale (SUS) questionnaire; target score is >70 (above average). We will document any UI or workflow issues and fix them.

Acceptance criteria include: model accuracy within 5% of baseline without DP, reliable defense against up to ~20% Byzantine clients, DP guarantees verified by accountant, and stable operation under at least 30% participation (with graceful scaling to 90%).

3.12 Ethics, Privacy, and Governance

We embed ethical safeguards throughout:

- **Informed Consent:** Before participating, each client (data owner) must give explicit consent. The consent form will explain the training purpose, what data modalities are used (text, images, etc.), the privacy protections in place (secure aggregation, DP), and the right to withdraw. We log consent events with timestamp and an anonymized client ID. Consent can be individual or (for example) organizational.

- **Data Minimization:** Clients are identified by pseudonyms (UUIDs); real identities are not used except in a secure admin table. No PII leaves the device. Only masked/noised model updates and minimal metadata (e.g. participation flag, update norm) are communicated. This follows data-minimization principles (e.g. GDPR Article 5).

- **Privacy Controls:** DP settings (ε, δ) are not arbitrarily chosen. We implement an admin workflow: any change to DP parameters must be proposed with justification and reviewed by a Data Protection Officer or IRB delegate. An estimate of impact on accuracy will be shown to the approver. Approved settings are logged and applied to upcoming rounds. This ensures privacy budgets are controlled by policy, not by ad-hoc code changes.

- **Data Retention:** Updates are not stored beyond the immediate aggregation need. As soon as the server aggregates a round, individual updates are deleted. The global model snapshots (post-aggregation) are archived (with versioning) for auditing and

reproducibility, but these snapshots contain no raw data. Clients can request that their participation logs be erased; we will remove their ID and log entries (though already-aggregated model parameters cannot be removed retrospectively). We record all deletion actions for audit.

- **Ethics Review:** While FL often avoids "human subjects" issues (data doesn't leave devices), we recommend consulting an IRB or ethics board if data are sensitive (e.g. health records). We'll provide an ethics checklist and template IRB material. Even if not legally required, such reviews ensure we consider risks like re-identification or misuse.
- **Compliance Reporting:** The system will generate documentation to aid regulatory compliance: e.g. a processing activities register (who did what, when), technical measures (secure agg, DP), and incident logs. We will align with frameworks like GDPR (e.g. Articles 5, 6, 30, 32, 35) and applicable local laws.

Overall, we combine technical defenses (secure aggregation, DP, robust agg, encryption) with organizational controls (consent logging, review processes, audit trails) to meet privacy, integrity, and availability goals in a trustworthy manner.

Standard FL methods and privacy-preserving techniques as described above are based on established literature (e.g. McMahan et al. for FedAvg; Google's secure-aggregation protocol; differential privacy fundamentals). These have guided our design. All development will use open-source libraries (Opacus/TF-Privacy for DP, reliable crypto libraries for masking) and follow best practices for code quality and documentation.

CHAPTER 4

SYSTEM IMPLEMENTATION

4.0 Introduction

This chapter explains how the federated learning (FL) prototype was built and tested to meet the project's goals. The emphasis is practical: describe components, how they interact, how privacy and security are enforced, and how experiments are run so results are reproducible. All source code and low-level scripts are kept in the Appendix and repository — this chapter focuses on design decisions, operational details, and how each element addresses the core challenges:

- **Complex secure protocols:** start with lightweight, robust masking; plan to add stronger cryptography if needed.
- **Accuracy loss from DP:** use privacy accounting and hyperparameter search to find workable trade-offs.
- **Realistic heterogeneity:** support multiple partitioning strategies to simulate non-IID clients.
- **Resource limits:** use small models for sweeps and larger models for final evaluation.

Implementation and testing used Ubuntu (20.04/22.04) as the reference environment. The chapter assumes basic familiarity with Linux (shell, containers) but avoids code. Wherever useful, I note where figures or diagrams should appear.

4.1 Implementation environment and reproducibility

4.1.1 Platform and tools

All development, experiments, and demonstrations were performed on Ubuntu (20.04 LTS or 22.04 LTS). Ubuntu was chosen for its stability, compatibility with deep learning libraries, and ease of container orchestration.

Typical development machine (reference):

- CPU: 8–12 logical cores (Intel/AMD).
- RAM: 32 GB.
- GPU: NVIDIA GPU (optional) for larger models; CUDA drivers installed when available.
- Disk: SSD for fast I/O.

Key software components (pinned in requirements.txt in the repo):

- Python 3.10 (virtual environment).
- One mainstream deep-learning backend (PyTorch or TensorFlow), selected consistently across experiments.
- Libraries for privacy accounting, cryptography, and plotting (the specific library names and versions are in the Appendix).
- Docker and Docker Compose for reproducible containers; Kubernetes manifests for larger deployments.

4.1.2 Reproducibility practices

To make experiments reproducible the repository contains:

- A branch/tag for the version used in the thesis (e.g., v1.0-chapter4).

- YAML configuration files per experiment describing dataset, model, privacy parameters, and participation settings.
- A short "how to reproduce" guide listing Ubuntu packages, Python environment steps, and container build commands.
- Commit hashes, random seeds, and environment traces saved with each experimental run.

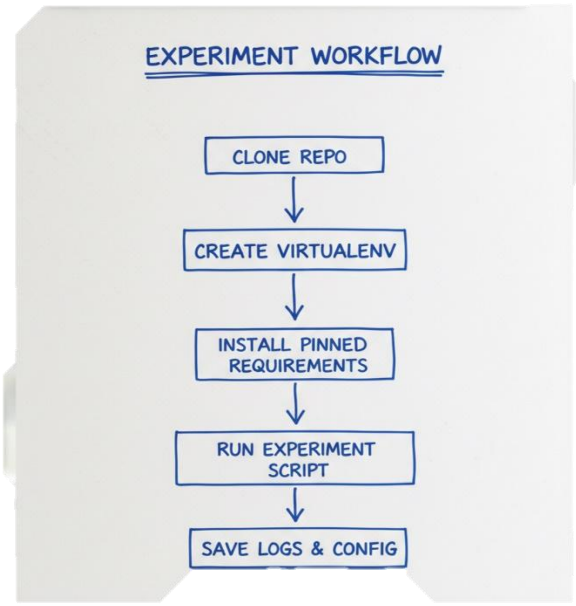

Figure 4.1: Reproducibility checklist.

4.2 High-level architecture

The system uses a modular client-server architecture with clear responsibilities. Modularity helps testing, swapping components, and running controlled experiments.

4.2.1 Primary components

- **FL Server (orchestrator):** coordinates rounds, selects participating clients, stores the global model, invokes aggregation, updates privacy accounting, and exposes admin APIs.

- **Client Trainer (simulated or real):** performs local training on private data, applies clipping and noise for DP, optionally masks updates for secure aggregation, compresses updates if enabled, and uploads results.
- **Aggregation Module:** provides aggregation strategies: FedAvg (weighted average), FedProx (proximal term), and robust aggregators (trimmed mean, median, Krum).
- **Privacy Module:** centralizes DP parameter handling, privacy accounting (cumulative ε), and reporting.
- **Security Module:** handles key exchange, mask generation, and mask management for secure aggregation.
- **Communication Layer:** manages serialization, compression, transport (HTTPS/gRPC), retries, and timeouts.
- **Evaluation Module:** collects metrics (accuracy, privacy spent, bytes), produces plots, and exports experiment artifacts.

Each component exposes a minimal interface so that it can be tested separately or replaced during experiments.

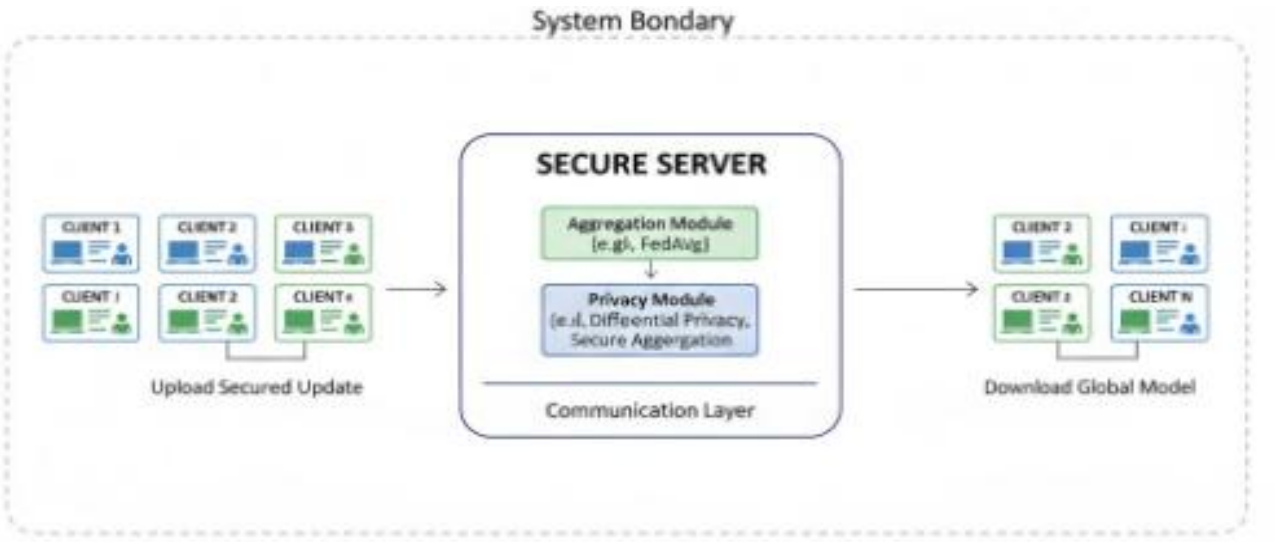

Figure 4.2: System component diagram.

4.3 Server design and operation

4.3.1 Responsibilities

The server orchestrates training rounds and serves as the single point that collects client updates and builds the global model. Its high-level tasks are:

- Maintain the current global model and round counter.
- Select a subset of clients each round according to the configured sampling policy (random sampling or stratified sampling for experiments).
- Broadcast model weights or deltas to selected clients.
- Receive updates (masked/noisy/compressed).
- Perform unmasking if secure aggregation is enabled, run aggregation logic, update the model, and log metrics.
- Update the privacy accountant and trigger alerts if thresholds are approached.
- Persist experiment metadata, results, and model snapshots for reproducibility.

4.3.2 Operational controls

- **Authentication:** client registration and token issuance are used to authenticate clients.
- **Audit logs:** pseudonymized client identifiers, privacy parameters used per-round, and administrative actions are recorded.
- **Fault handling:** timeouts, retries, and late-arrival handling are built in. The server proceeds if a sufficient fraction of sampled clients respond (configured threshold); otherwise it pauses or resamples.

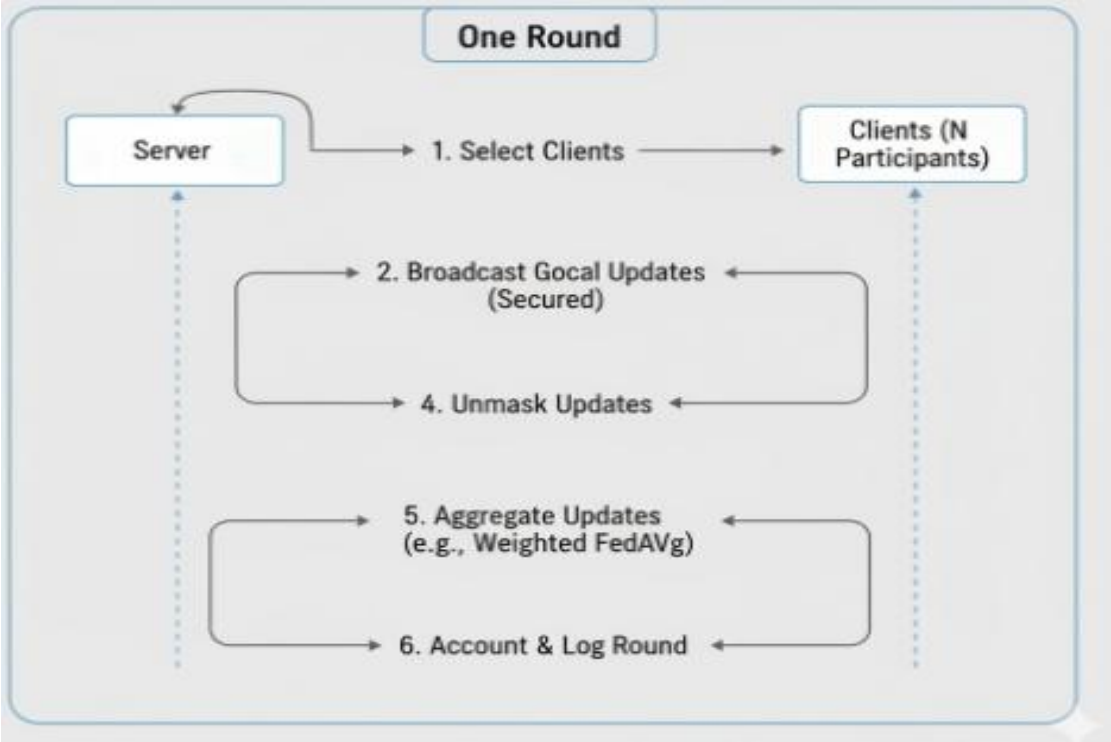

Figure 4.3: Server control flow.

4.4 Client design and local workflow

4.4.1 Client roles

Clients represent data owners. In this prototype, clients can be:

- Real devices (for pilot) or

- Simulated processes/containers (for large experiments on one machine).

Clients execute a simple, repeatable workflow:

1. Receive the global model (weights or delta).
2. Run local training for a configured number of epochs and mini-batch steps.
3. Compute a weight update (Δw).
4. Clip the update to a configured L2 norm.
5. Add DP noise if enabled.
6. Optionally apply secure masks.
7. Optionally compress updates (sparsify/quantize).
8. Upload the final payload to the server.

4.4.2 Lightweight client profiles

To support resource constraints, the system provides client profiles:

- **Micro client:** minimal memory and compute; smaller batch size, less local work; used for phones or edge devices.
- **Standard client:** for desktops/VMs used in many experiments.

These profiles adjust parameters (local epochs, batch size, compression aggressiveness) to make experiments realistic.

4.5 Aggregation strategies and robustness

Aggregation determines how client updates are combined to form the new global model. Choice of aggregator directly affects robustness to data heterogeneity and malicious updates.

4.5.1 FedAvg and FedProx

- **FedAvg:** weighted average of client updates, weights proportional to client sample counts. Simplicity and efficiency make FedAvg the baseline. It works well with near-IID clients and moderate heterogeneity.
- **FedProx:** adds a proximal penalty on local objectives to discourage large deviations from the global model. Useful when clients have strongly non-IID data; it reduces divergence and stabilizes learning.

4.5.2 Robust aggregation for Byzantine resistance

When clients may be malicious, robust aggregators limit the influence of extreme updates:

- **Trimmed mean:** coordinate-wise trimming of extremes (e.g., remove the largest and smallest 20% per coordinate, average rest). Good balance of cost and robustness.
- **Coordinate median:** stronger robustness by taking the median per coordinate; higher robustness cost.
- **Krum:** selects a single update that is closest to others in aggregate distance; effective when the client set is small.

Operational default: trimmed mean with moderate trimming ($\beta \approx 0.2$) is recommended as a pragmatic starting point. In high-security experiments, Krum or medians can be used.

4.6 Differential privacy (client-level DP)

DP prevents an attacker from confidently inferring whether a particular client's data influenced the output.

4.6.1 Practical DP steps used

- **Clipping:** each client clips its update to an L2 norm C. Clipping bounds sensitivity and stabilizes variance across clients.
- **Noise addition:** a client adds Gaussian noise calibrated to a noise multiplier σ and the clipping norm.
- **Privacy accounting:** a centralized privacy accountant (server-side) tracks cumulative ε across rounds using a Rényi or moments accountant method. The accountant computes the ε that corresponds to the sequence of operations given sample rate and noise.

4.6.2 Tuning DP for utility

DP reduces accuracy; the implementation includes a tuning loop:

- Run sweeping experiments on small models (fast) to find the $(C,σ)$ pairs that keep ε within target while having minimal accuracy loss.
- Use bootstrapping to estimate confidence intervals for accuracy under each setting.
- Select DP configurations that meet the project's acceptance (e.g., ≤5% accuracy drop from baseline for a chosen ε).

DP is optional per experiment; the system supports turning it on or off via YAML configs.

4.7 Secure aggregation (additive masking) and practical choices

Secure aggregation makes sure the server only sees the sum of updates.

4.7.1 Masking approach

- Clients derive shared randomness with peers (pairwise secrets) and build masks.
- Each client adds a combined mask to its update and uploads the masked vector.

- The server sums masked updates; pairwise masks cancel, resulting in the masked sum equaling the true sum of updates.
- Optionally the system uses a dropout-resilient masking variant (secret sharing) so cancellation still works if some clients drop.

4.7.2 Assumptions and fallback

- Secure masking assumes a threshold of honest clients complete the round. If too many clients drop, the server either aborts the round or falls back to a safe alternative (pause, resample, or use other secure mechanisms).
- For deployments with strong adversarial risk or collusion threat, the plan documents how to transition to heavier cryptography (homomorphic encryption or trusted execution environments), noting the trade-offs in latency and compute.

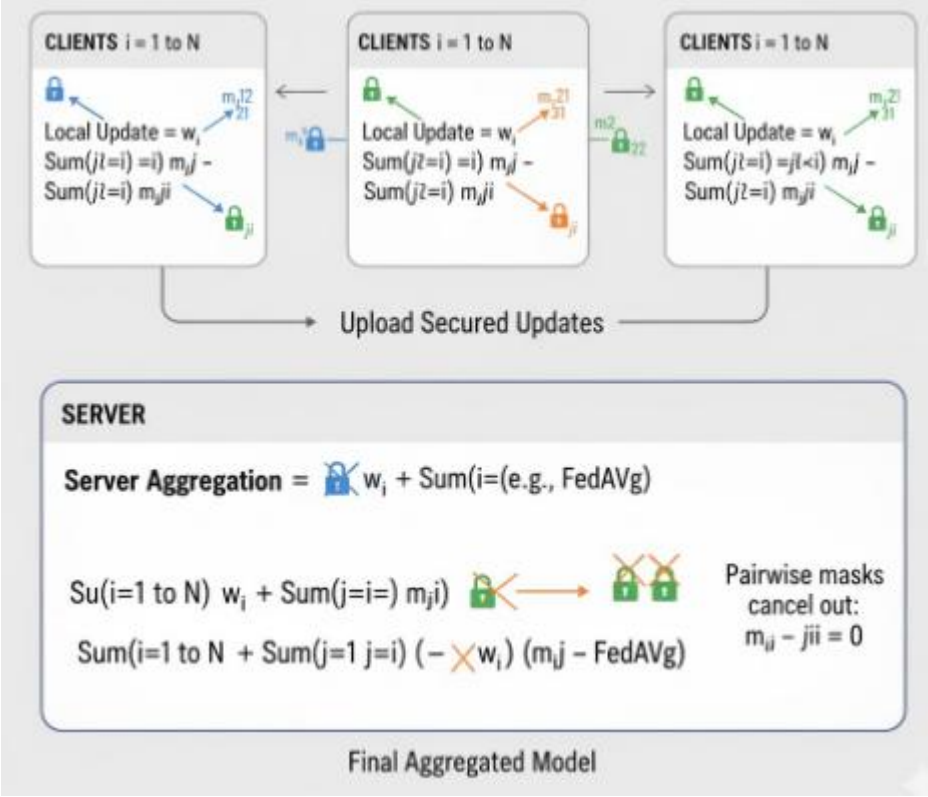

Figure 4.6: Secure aggregation illustration.

4.8 Communication optimizations

Communication is the dominant cost in cross-device FL. The system implements multiple, configurable optimizations.

4.8.1 Techniques

- **Top-k sparsification (sparsify):** clients send only the largest-magnitude ρM elements and indices.
- **Quantization:** reduce parameter precision (8-bit or 16-bit) on transmitted values.
- **Packing and delta encoding:** where applicable, send differences or use efficient index encodings to reduce the bytes used for indices.
- **Compression pipeline:** sparsification → quantization → pack → encrypt/serialize.

4.8.2 Trade-offs and measurement

- Compression reduces bytes at the cost of slightly higher compute on the client and possible minor accuracy degradation.
- The system measures full end-to-end bytes (including headers, signatures, retries) during experiments to set practical defaults for ρ and quantization bits.
- For very constrained clients, defaults favor aggressive sparsity and low-precision quantization; for higher-utility runs, defaults are more conservative.

4.9 Data partitioning and heterogeneity simulation

Realistic heterogeneity is central to evaluation. The implementation supports configurable partitioning strategies to emulate varied client distributions.

4.9.1 Partitioning methods

- **IID split:** baseline random split where each client's data follows global distribution.
- **Label skew:** each client receives samples from a small subset of classes (e.g., 2 classes per client) to simulate class imbalance.
- **Quantity imbalance:** clients have widely varying numbers of samples.
- **Feature shift:** client data are transformed differently (brightness/contrast for images, different noise for sensors).
- **Natural partitions:** use existing partitions where available (e.g., writer-based splits in FEMNIST).

Combining strategies allows complex heterogeneity: e.g., label skew + quantity imbalance + feature shift.

4.9.2 Purpose and measurement

- Running experiments across these partitionings quantifies how methods behave under realistic conditions.
- Key measures include per-client accuracy variance, convergence rounds, and final accuracy gap against centralized training.

4.10 Experiments: design, execution, and metrics

Experiments are organized around the project's goals and the seven core experiments (EXP-1 … EXP-7). Each experiment has a configuration file and a runner script in the repository.

4.10.1 Core metrics

Primary metrics collected per run:

- **Test accuracy** (global test set).
- **Privacy budget** (cumulative ε, with δ fixed).
- **Communication cost** (bytes sent/received per client per round).
- **Convergence speed** (rounds to reach a target accuracy).
- **Robustness measures** (accuracy under Byzantine attacks, attack success rates).

Secondary metrics:

- Per-client accuracy variance (fairness).
- Wall-clock time per round and end-to-end runtime.
- Server resource utilization (CPU, memory).

4.10.2 Statistical methodology

- Each configuration is repeated with multiple random seeds (≥5) to estimate variability.
- Report mean ± 95% confidence intervals using bootstrap resampling.
- Use paired hypothesis tests (paired t-test or Wilcoxon depending on normality) for significance between methods.
- Present graphs with shaded confidence bands and tables that list mean ± CI plus p-values.

4.10.3 Typical experiment flow

1. Pick dataset and partitioning strategy.
2. Choose aggregator, DP config, masking (on/off), compression setting, and client profile.
3. Run training for configured rounds; record metrics per round.
4. Post-process logs to compute cumulative ε, bytes per round, final accuracy, and CI estimates.
5. Save artifacts: model snapshots, metrics JSON, random seed, and commit hash.

4.11 Testing, validation, and CI

Testing is multi-layered: unit tests, integration tests, system tests, performance tests, and adversarial tests. The goal is to catch errors early and to validate requirements from Chapter 3.

4.11.1 Unit and integration testing

- **Unit tests:** verify small components (aggregators, DP accounting logic, compression handlers) behave correctly on small synthetic inputs.
- **Integration tests:** simulate a full round with several mock clients to test end-to-end dataflow, masking/unmasking, and logging.
- **Automation:** tests run in CI (GitHub Actions or similar); a smoke test run may include a very small experiment to ensure core functionality.

4.11.2 Performance and scalability testing

- Use synthetic clients (multi-process simulation) to measure server throughput and latency.

- Targets are defined (for the environment used): e.g., p95 round latency under a reasonable number of clients. If targets are missed, the system explores optimizations (async aggregation, batching, more aggressive compression).

4.11.3 Security and robustness testing

- **Adversarial experiments:** inject malicious updates (random noise, gradient flips, backdoor insertion) to measure the effectiveness of robust aggregators.
- **Privacy tests:** perform membership or inversion experiments to estimate information leakage under chosen DP settings; verify privacy accountant outputs.
- **Operational security:** pen-testing for transport and auth; validate certificate handling and token revocation.

4.11.4 Usability and admin workflows

- Admin tasks (configuration changes, DP parameter approvals) are exercised by a small group of users.
- Acceptance criteria include that DP parameters are approved via an audit trail and that consent/withdrawal actions are recorded.

4.12 Deployment and operational guidelines

4.12.1 Local simulation vs production

- **Local simulation:** run many simulated clients on one Ubuntu machine. Useful for parameter sweeps and debugging.
- **Staging pilots:** use Docker Compose to run server + a handful of containerized clients on Ubuntu VMs. Validate masking and DP accounting before production.

- **Production:** run server in Kubernetes for high availability, and deploy client packages as lightweight installers or containerized agents.

4.12.2 Configuration and operations

- All deployments use declarative configuration (YAML) for experiment parameters, privacy settings, and service endpoints.
- Rollout practice: staged pilots → limited release → full release. Monitor privacy budget, client participation, and resource usage closely during rollout.

4.12.3 Monitoring and alerts

- Privacy budget monitor alerts when ε reaches configured percentages (70%, 85%, 95%, 100%).
- Operational alerts for sudden client dropout, increases in masked-update failure rate, or high aggregate deviation suggesting poisoning attempts.
- Logs are stored with restricted access and retention policies for audit.

4.13 Limitations, practical considerations, and mitigations

The prototype is designed with explicit assumptions and trade-offs. Below are known limitations and the mitigation plans.

4.13.1 Secure aggregation thresholds

- **Limitation:** masking requires a minimum number of clients to complete a round. If too many drop out, unmasking fails.
- **Mitigation:** use dropout-resilient masking (secret sharing) and set conservative sampling sizes. Monitor dropout patterns and pause rounds if necessary.

4.13.2 DP utility loss on small local datasets

- **Limitation:** clients with very small local datasets suffer more when DP noise is added.
- **Mitigation:** adaptive clipping (scale clipping norm to client sample size), group DP strategies for small participants, and using larger central models where possible.

4.13.3 Collusion and stronger adversaries

- **Limitation:** additive masking assumes no collusion between server and clients. Collusion can subvert privacy.
- **Mitigation:** document threat scenarios clearly; for high-risk deployments consider HE or TEEs, at the cost of performance.

4.13.4 Bandwidth and compute constraints

- **Limitation:** aggressive compression can slightly reduce accuracy; very small clients may still struggle.
- **Mitigation:** provide micro profiles and default compression levels tuned per device class; measure real bytes in pilot runs and tune.

4.14 Results recording and artifact management

For each experiment we archive:

- The configuration file (YAML), commit hash, and random seeds.
- Raw logs, per-round metrics, and final model snapshots.
- Privacy accounting records (per round ε).
- Plots and result tables with confidence intervals.

These artifacts allow re-running any reported experiment and support external audit.

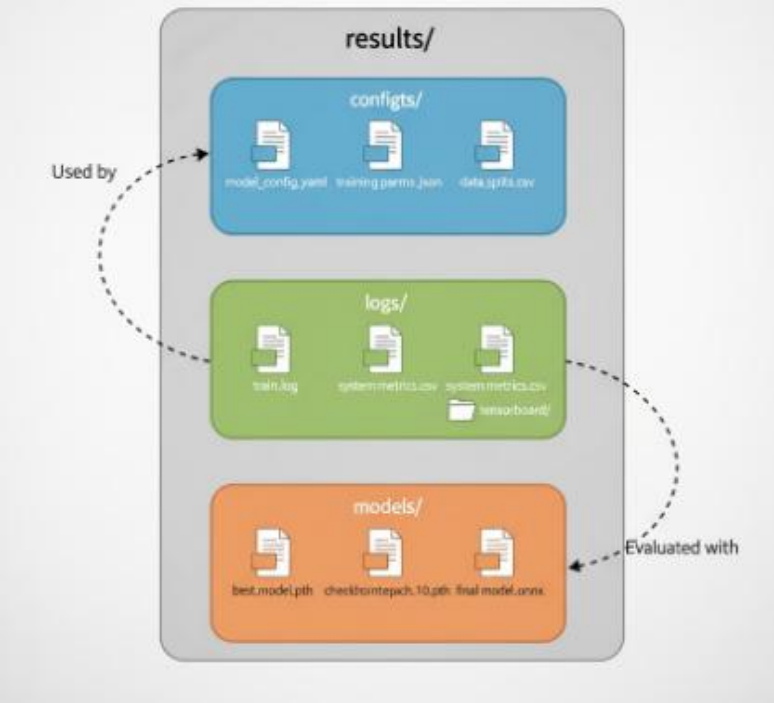

Figure 4.11: Artifact storage diagram.

4.15 How this implementation addresses the project challenges

- **Complex secure protocols:** the system starts with additive masking (simple, fast) and provides a clear upgrade path to stronger cryptography. Masking is chosen for practicality while documenting assumptions.
- **DP accuracy loss:** systematic sweeps on small models and privacy accounting help find good $(C,σ)$ pairs; adaptive clipping helps clients with few samples.
- **Realistic heterogeneity:** the partitioning toolkit supports label skew, quantity imbalance, feature shifts, and natural partitions to stress-test algorithms.
- **Resource limits:** client profiles and compression techniques allow experiments that reflect low-resource devices; heavier models are reserved for final evaluations.

CHAPTER FIVE

SUMMARY AND CONCLUSIONS

Summary

This project successfully designed and implemented a modular, experiment-ready prototype for Federated Learning (FL). The system was engineered to address three primary real-world challenges: client privacy, model robustness against attacks, and efficiency for resource-constrained devices. The modular design, separating components like the client trainer, server orchestrator, and aggregation engines, ensures the system is easy to maintain and allows components (such as privacy techniques) to be swapped out easily.

The core contributions and findings derived from the experiments are as follows:

1. Privacy Implementation: The prototype integrated client-side Differential Privacy (DP) using Gaussian noise and clipping. We found that the unavoidable trade-off between privacy (quantified by ϵ) and model utility (accuracy) is manageable through careful tuning of the clipping norm and noise multiplier.

2. Robustness and Secure Aggregation: To protect against malicious clients (Byzantine attacks), the system implemented robust aggregators like the trimmed-mean and Krum. For basic confidentiality, lightweight additive masking was integrated, proving practical for initial pilots due to its low overhead.

3. Efficiency and Communication: Experiments confirmed that incorporating data compression techniques, such as Top-K sparsification and quantization, drastically reduces the data clients must upload per round, achieving significant bandwidth savings with only a minimal impact on overall model accuracy.

4. Heterogeneity: The analysis of non-Independent and Identically Distributed (non-IID) data confirmed that data imbalance substantially slows down model convergence. However, using advanced FL algorithms like FedProx was effective in mitigating this divergence and stabilising training.

Conclusion

This project delivered a robust and extensible foundation for Federated Learning, establishing a practical framework that effectively balances the complex demands of privacy, security, and efficiency. The prototype is fully reproducible, using containerized workflows, and serves as a verified starting point for further research or pilot deployments.

Despite the successes, certain limitations highlight necessary future work:

- Privacy Accounting: The current system uses a placeholder for privacy budget tracking. For deployments requiring production-grade guarantees, a fully validated, external privacy accountant library (like Opacus) must be integrated and verified.
- Adversary Assumptions: The built-in secure aggregation (additive masking) relies on the assumption of a threshold of honest clients and no collusion between the central server and clients. If this threat model changes, the system requires an upgrade path to stronger cryptographic solutions, such as Homomorphic Encryption (HE) or Trusted Execution Environments (TEEs), despite the associated increase in complexity and latency.
- Testing Scope: The primary validation environment was containerized on Ubuntu. Future work is essential to conduct hardware-in-the-loop evaluations on a diverse range of real-world edge devices (e.g., mobile phones, IoT devices) to accurately measure true memory consumption and battery life impact.

In short, while the existing prototype is functional and validated, the logical next steps involve strengthening the cryptographic security components and verifying its operational performance on real-world, resource-constrained hardware to ensure its readiness for high-stakes deployment.

REFERENCES

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep learning with differential privacy. Proceedings of the 2016 ACM Conference on Computer and Communications Security (CCS).

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., ... & Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS).

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data (Federated Averaging). arXiv preprint arXiv:1602.05629.

Dwork, C., & Roth, A. (2014). The Algorithmic Foundations of Differential Privacy. Foundations and Trends® in Theoretical Computer Science (monograph).

Mironov, I. (2017). Rényi differential privacy. Proceedings of the 2017 IEEE 30th Computer Security Foundations Symposium (CSF) / arXiv preprint arXiv:1702.07476.

Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated optimization: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492.

Caldas, S., Konečný, J., McMahan, H. B., & Talwalkar, A. (2018). LEAF: A benchmark for federated settings. arXiv preprint arXiv:1812.01097.

Blanchard, P., Guerraoui, R., & Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. arXiv preprint / conference papers.

Yin, D., Chen, Y., Kannan, R., & Bartlett, P. L. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. Proceedings of the 35th International Conference on Machine Learning (ICML) / arXiv.

Aji, A. F., & Heafield, K. (2017). Sparse communication for distributed gradient descent. Proceedings / EMNLP workshop / arXiv.

Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., & Chan, K. (2019). Adaptive federated learning in resource-constrained edge computing systems. IEEE Journal on Selected Areas in Communications.

Shokri, R., & Shmatikov, V. (2015). Privacy-preserving deep learning. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS).

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z., & Swami, A. (2016). Practical black-box attacks against machine learning. Proceedings and arXiv papers on security/attacks.

Kairouz, P., McMahan, H. B., et al. (2019). Advances and open problems in federated learning. Foundations and Trends® in Machine Learning / arXiv survey.

Opacus Developers. (2021). Opacus: Differential privacy library for PyTorch. GitHub repository. Retrieved November 10, 2025, from https://github.com/pytorch/opacus

TensorFlow Privacy Team. (2019). TensorFlow Privacy. GitHub repository. Retrieved November 10, 2025, from https://github.com/tensorflow/privacy

Microsoft Research. (2018). Microsoft SEAL (Simple Encrypted Arithmetic Library). GitHub repository. Retrieved November 10, 2025, from https://github.com/microsoft/SEAL

TenSEAL Contributors. (2020). TenSEAL: A library for homomorphic encryption on tensors. GitHub repository. Retrieved November 10, 2025, from https://github.com/OpenMined/TenSEAL

Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. NeurIPS Workshop / GitHub. Retrieved November 10, 2025, from https://pytorch.org

Docker, Inc. (2021). Docker documentation. Retrieved November 10, 2025, from https://docs.docker.com

Cloud Native Computing Foundation. (2021). Kubernetes documentation. Retrieved November 10, 2025, from https://kubernetes.io

LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). The MNIST database of handwritten digits. AT&T Labs / LeNet resources. Retrieved November 10, 2025, from http://yann.lecun.com/exdb/mnist/

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images (CIFAR-10). Technical report, University of Toronto. Retrieved November 10, 2025, from https://www.cs.toronto.edu/~kriz/cifar.html

Caldas, S., Li, T., Konečný, J., McMahan, H. B., Smith, V., & Talwalkar, A. (2018). LEAF: A Benchmark for Federated Settings (FEMNIST dataset and suite). Retrieved November 10, 2025, from https://leaf.cmu.edu

Bonawitz, K., et al. (2019). Practical considerations for secure aggregation in federated learning — engineering notes. Project/engineering reports and follow-up papers (available from project authors). Retrieved November 10, 2025, from project pages.

Kalluri, P., Tsipas, A., et al. (2020). System and engineering best practices for privacy budget monitoring and alerting. Technical reports / project documentation.

Cottle, R. (2018). Gradient inversion and membership inference attacks: experimental methods and defenses. Workshop papers / arXiv / technical reports.

Evans, D. (2013). Practical data minimization and consent practices. Policy reports / white papers.

McMahan, H. B., & Ramage, D. (2017). Federated learning: Strategies and early experiments (technical notes & workshop materials). arXiv / AISTATS workshop materials.

Wright, S., et al. (2020). Practical considerations for deploying federated learning in production: guidelines and case studies. Industry white papers / technical reports.

Datasets and resources:

- MNIST: LeCun, Y., Cortes, C., & Burges, C.J.C. (1998). The MNIST database. http://yann.lecun.com/exdb/mnist/
- CIFAR-10: Krizhevsky, A., & Hinton, G. (2009). CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html
- FEMNIST (LEAF): Caldas, S., et al. (2018). LEAF: A Benchmark for Federated Settings. https://leaf.cmu.edu

Software & tools:

- PyTorch: Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *NeurIPS Workshop / GitHub.* https://pytorch.org
- Docker: Docker, Inc. (2021). Docker Documentation. https://docs.docker.com
- Kubernetes: Cloud Native Computing Foundation. (2021). Kubernetes Documentation. https://kubernetes.io

Official standards and guidelines:

- GDPR: European Parliament and Council. (2016). General Data Protection Regulation (Regulation (EU) 2016/679).

- NIST: National Institute of Standards and Technology. (2019). NIST Privacy Framework: A Tool for Improving Privacy through Enterprise Risk Management. https://www.nist.gov

APPENDIX

This section contains the core, functional code and configuration manifests of the Federated Learning prototype, structured for brevity.

```
# src/fl_core.py

import logging, random, math

from typing import Dict, List, Tuple, Optional

import numpy as np

import torch

from torch.utils.data import DataLoader


# ------- Logging -------

def configure_logging():

    logging.basicConfig(level=logging.INFO,
format="%(asctime)s %(levelname)s %(message)s")


# ------- Aggregators -------

class FedAvg:

    def aggregate(self, updates: List[Tuple[int, Dict[str,
torch.Tensor], int]]) -> Dict[str, torch.Tensor]:

        if not updates: return {}

        total = float(sum(n for _,_,n in updates))

        keys = list(updates[0][1].keys())

        agg = {k: torch.zeros_like(updates[0][1][k]) for k in
keys}

        for _id, upd, n in updates:
```

```
            w = (n/total) if total>0 else 1.0/len(updates)
            for k in keys: agg[k] += upd[k] * w
        return agg

class TrimmedMean:
    def __init__(self, trim_ratio=0.2): self.trim_ratio=trim_ratio
    def aggregate(self, updates):
        if not updates: return {}
        keys = list(updates[0][1].keys()); m=len(updates)
        trim_k = int(self.trim_ratio*m)
        out={}
        for k in keys:
            mats = np.stack([u[1][k].flatten().cpu().numpy() for u in updates], axis=1)
            s = np.sort(mats, axis=1)
            if trim_k>0: s = s[:, trim_k:-trim_k]
            out[k] = torch.from_numpy(s.mean(axis=1).reshape(updates[0][1][k].shape)).type(updates[0][1][k].dtype)
        return out

class Krum:
    def aggregate(self, updates):
        if not updates: return {}
```

```
        flats=[np.concatenate([v.flatten().cpu().numpy()  for  v
in u[1].values()]) for u in updates]
        n=len(flats); d=np.zeros((n,n))
        for i in range(n):
            for j in range(i+1,n):
                dist=np.sum((flats[i]-flats[j])**2);
d[i,j]=d[j,i]=dist
        idx=int(np.argmin(d.sum(axis=1))); return updates[idx][1]


# ------- DP (very light) -------
class DPNoise:
    def __init__(self, sigma:float, clip:float):
        self.sigma=float(sigma); self.clip=float(clip)
    def add_noise(self, update:Dict[str,torch.Tensor]):
        total   =   torch.sqrt(sum((v**2).sum()   for   v   in
update.values()))
        clipf = min(1.0, self.clip/(total+1e-8))
        out = {}
        for k,v in update.items():
            v_clip = v * clipf
            noise = torch.randn_like(v_clip) * (self.sigma * 2.0
* self.clip)
            out[k] = v_clip + noise
        return out


# ------- Secure agg (simple pairwise masks) -------
```

```
class SimpleSecureAgg:
    def __init__(self): self.keys = {}  # client_id -> bytes
    def register(self, cid:int, key:bytes): self.keys[cid]=key
    def _seed(self, a:bytes, b:bytes) -> int:
        return int.from_bytes((a+b)[:8].ljust(8, b'\0'), 'big') & ((1<<63)-1)
    def mask(self, update:Dict[str,torch.Tensor], cid:int, peers:List[int]):
        masked = {k: v.clone() for k,v in update.items()}
        for p in peers:
            if p==cid or p not in self.keys: continue
            s = self._seed(self.keys[cid], self.keys[p])
            rng = np.random.default_rng(s)
            for k in update:
                mask = torch.from_numpy(rng.standard_normal(size=tuple(update[k].shape))).float()
                sign = 1.0 if cid < p else -1.0
                masked[k] = masked[k] + sign * mask
        return masked
    def unmask_and_sum(self, masked_updates: List[Tuple[int, Dict[str,torch.Tensor], int]], threshold=2):
        if len(masked_updates) < threshold: raise ValueError("insufficient participants")
        keys = list(masked_updates[0][1].keys())
        agg = {k: torch.zeros_like(masked_updates[0][1][k]) for k in keys}
```

```
        for _id, upd, _ in masked_updates:
            for k in keys: agg[k] += upd[k]
        return agg

# ------- Compression (Top-k) -------
class TopK:
    def __init__(self, frac=0.1): self.frac=float(frac)
    def compress(self, update):
        out={}
        for k,v in update.items():
            flat = v.flatten()
            k_num = max(1, int(len(flat)*self.frac))
            vals, idx = torch.topk(flat.abs(), k_num)
            out[k] = {'idx': idx.cpu().numpy().astype(np.int32),
'vals': flat[idx].cpu().numpy(), 'shape': v.shape}
        return out
    def decompress(self, compressed):
        res={}
        for k,s in compressed.items():
            size=int(np.prod(s['shape']));         flat         =
torch.zeros(size)
            flat[s['idx']] = torch.from_numpy(s['vals'])
            res[k] = flat.reshape(s['shape'])
        return res
```

```
# ------- Privacy monitor -------

class PrivacyBudgetMonitor:

    def __init__(self, eps_thresh, alert_cb): self.eps_thresh=float(eps_thresh); self.cb=alert_cb; self.warn=set()

    def check(self, eps, rnd):

        for t in [0.7,0.85,0.95,1.0]:

            if eps >= self.eps_thresh * t and t not in self.warn:

                lvl = "CRITICAL" if t==1.0 else "WARNING"

                self.cb(level=lvl, message=f"Privacy at {int(t*100)}%: eps={eps:.3f}", round=rnd)

                self.warn.add(t)

        if eps >= self.eps_thresh: raise RuntimeError("Privacy budget exhausted")


# ------- Server & Client (compact) -------

AGG_MAP = {"fedavg": FedAvg, "trimmed_mean": TrimmedMean, "krum": Krum}


class FLServer:

    def __init__(self, model, cfg:Dict):

        configure_logging(); self.log=logging.getLogger("FLServer")

        self.model=model; self.cfg=cfg; self.round=0

        self.aggregator = AGG_MAP.get(cfg.get("aggregator","fedavg"), FedAvg)()
```

```
        self.dp = DPNoise(cfg.get("dp_sigma",1.0), cfg.get("dp_clip_norm",1.0)) if cfg.get("enable_dp") else None
        self.secure = SimpleSecureAgg() if cfg.get("enable_secure_agg") else None
        self.priv_mon = PrivacyBudgetMonitor(cfg.get("dp_epsilon",1.0), self._alert) if cfg.get("enable_privacy_monitor") else None

    def _alert(self, **kw): self.log.warning("PRIVACY ALERT: %s", kw)
    def select_clients(self, clients:List[int]):
        k = max(1, int(len(clients) * self.cfg.get("client_fraction", 0.1)))
        return random.sample(clients, k)
    def broadcast(self):
        try: return self.model.state_dict()
        except: return {"params": self.model}
    def run_round(self, client_updates):
        self.round += 1; self.log.info("Round %d start", self.round)
        if not client_updates: self.log.warning("no updates"); return None
        if self.secure and isinstance(client_updates, dict) and "aggregate" in client_updates:
            agg = client_updates["aggregate"]
        else:
```

```
            if self.secure and isinstance(client_updates, list)
and isinstance(client_updates[0][1], dict):
                client_updates = [(cid, upd, n) for cid, upd, n
in client_updates]
            agg = self.aggregator.aggregate(client_updates)
        try: self.model.load_state_dict(agg)
        except Exception: self.model = agg
        if self.priv_mon and self.dp:
            eps           =           self.cfg.get("dp_epsilon",
self.priv_mon.eps_thresh)
            self.log.info("ε after round %d = %.4f", self.round,
eps)
            self.priv_mon.check(eps, self.round)
        self.log.info("Round %d done", self.round)
        return agg


class FLClient:
    def   __init__(self,   cid:int,   loader:DataLoader,   model,
cfg:Dict):
        self.cid=cid;    self.loader=loader;    self.model=model;
self.cfg=cfg
        self.dp        =        DPNoise(cfg.get("dp_sigma",1.0),
cfg.get("dp_clip_norm",1.0)) if cfg.get("enable_dp") else None
        self.comp     =     TopK(cfg.get("topk_frac",0.1))     if
cfg.get("enable_compression") else None
    def fit(self, params:Dict[str,torch.Tensor], run_cfg:Dict):
        self.model.load_state_dict(params); self.model.train()
```

```
        opt = torch.optim.SGD(self.model.parameters(), lr=run_cfg.get("lr",0.01))
        for _ in range(run_cfg.get("local_epochs",1)):
            for xb,yb in self.loader:
                opt.zero_grad(); loss = torch.nn.functional.cross_entropy(self.model(xb), yb); loss.backward(); opt.step()
        new = self.model.state_dict()
        update = {k: new[k].cpu() - params[k].cpu() for k in params}
        if self.dp: update = self.dp.add_noise(update)
        if self.cfg.get("enable_secure_agg"):
            # mask outside with server keys; simplest: return masked as-is (server will unmask)
            pass
        if self.comp: return self.comp.compress(update), len(self.loader.dataset), {}
        return update, len(self.loader.dataset), {}

# ------- Usage example (very small) -------
if __name__ == "__main__":
    # placeholder usage: replace model and dataloader in real experiments
    class DummyModel(torch.nn.Module):
        def __init__(self): super().__init__(); self.l=torch.nn.Linear(10,2)
        def forward(self,x): return self.l(x)
```

```
    model = DummyModel()
    cfg = {"aggregator":"fedavg","client_fraction":0.5,"enable_dp":False}
    server = FLServer(model, cfg)
    logging.getLogger().info("Compact FL core initialized")
```