Balancing Privacy and Performance in Federated Learning with Adaptive Differential Privacy and Secure Multi Party Computation

T.D Attygalle

Balancing Privacy and Performance in Federated Learning with Adaptive Differential Privacy and Secure Multi Party Computation

T.D Attygalle

Index No: **20000172**

Supervisor: Dr. Ajantha Athukorala

May 2025

Submitted in partial fulfillment of the requirements of the B.Sc in Computer Science Final Year Project (SCS4224)

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: T.D Attygallle

[illegible]

Signature of Candidate: **Date:** 24/06/2025

This is to certify that this dissertation is based on the work of

Mr.T.D Attygallle

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor's Name: Dr.Ajantha Athukorala

[illegible]

Signature of Supervisor: **Date:** 24/06/2025

Abstract

Federated Learning (FL) enables multiple clients to collaboratively train deep models without sharing their raw data, yet model updates can still leak sensitive information through inference attacks. Differential Privacy (DP) offers formal privacy guarantees by perturbing updates with noise, but naively adding noise at each client often degrades model utility and rapidly consumes the privacy budget. Adaptive DP-FL developed by Fu et al. (2022) mitigates this by using per-client adaptive clipping and validation-driven noise decay, but still relies on each client injecting noise locally, increasing variance and requiring trust in the aggregation server. Secure Multi-Party Computation (SMPC) via Shamir's secret sharing ensures that individual updates remain hidden during aggregation, but has not been combined with adaptive DP in FL.

In this research, we propose a novel hybrid FL framework that integrates adaptive DP-FL with SMPC. Each client adaptively clips its gradients and secret-shares them across multiple non-colluding aggregation servers. The servers securely aggregate the shares, reconstruct only the global sum, and inject a single, globally calibrated Gaussian noise term to satisfy (ε, δ)-DP under Rényi DP accounting of Mironov (2017). This design reduces the effective sensitivity by a factor of $1/K$ (with K clients), lowers total noise variance by $O(K^2)$, and simplifies privacy bookkeeping.

We implement both the baseline adaptive DP-FL and our hybrid scheme in PyTorch using Opacus, and evaluate them on MNIST and Fashion-MNIST under privacy budgets $\varepsilon = 0.5$ and $\varepsilon = 0.3$. The hybrid method achieves up to 96.45% and 94.32% test accuracy on MNIST versus 95.89% and 94.27% for the baseline, and 79.27% and 78.42% on Fashion-MNIST versus 79.00% and 76.21%, respectively. Accuracy and loss curves against ε demonstrate tighter privacy-utility trade-offs. Our results show that combining adaptive DP with SMPC offers a practical path to stronger privacy guarantees and improved model performance, making FL more viable for sensitive domains such as healthcare and finance.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Ajantha Athukorala, for his invaluable guidance, encouragement, and patient mentorship throughout every stage of this research. His insightful feedback and unwavering support have been instrumental in shaping both the technical and conceptual aspects of this thesis.

I am also profoundly grateful to Dr. Asanka Sayakkara, whose timely advice and problem-solving assistance helped me overcome numerous technical blockers. His willingness to discuss ideas at length and suggest practical solutions was crucial to keeping this project on track.

My heartfelt thanks go to my parents, whose love, understanding, and financial support provided me with the stability and confidence to pursue this work. Their constant encouragement has been a source of strength and motivation.

Finally, I gratefully acknowledge my friends, whose thoughtful discussions, and occasional diversion helped me maintain a healthy balance between work and life. Their support made the journey of completing this thesis all the more rewarding.

Table of Contents

1 Introduction **1**

1.1 Background to the Research 1

1.2 Problem Statement 3

1.3 Research aim, questions and objectives 3

1.3.1 Research aim 3

1.3.2 Research questions 4

1.3.3 Research objectives 4

1.4 Significance of the Project 5

1.5 Research Approach and Methodology 5

1.6 Outline of the Dissertation 6

1.7 Delimitation of Scope 8

1.7.1 In Scope 8

1.7.2 Out Of Scope 8

2 Background & Literature review **9**

2.1 Training Process 9

2.2 Key Challenges 9

2.2.1 Privacy Challenges in Federated Learning 10

2.3 Privacy Preservation in Federated Learning 11

2.3.1 Differential Privacy 11

2.3.2 Secure Multi-Party Computation (SMPC) and Secret Sharing . . 12

2.4 Balancing Privacy and Utility in Federated Learning 14

2.5 Research Gap 15

3 Design **17**

3.1 High-Level Framework Overview 17

3.1.1 Interaction of Components 17

3.1.2 Pipeline Flowchart 18

3.2 Baseline System Architecture 19

3.2.1 Workflow 19

3.3 Proposed Hybrid Architecture (Adap DP-FL + SMPC) 23

3.3.1 Client-Side Process 23

3.3.2 Server-Side Process 24

3.3.3 Privacy Accounting 25

3.4 Final Proposed Model's Algorithm . . . 29

4 Implementation **30**

4.1 Development Setup . . . 30

4.1.1 Programming Languages and Libraries . . . 30

4.1.2 Hardware Environment . . . 30

4.2 Dataset Preparation . . . 30

4.3 Model Architecture . . . 32

4.4 Adaptive Clipping, Adaptive Noise, and Local Training . . . 33

4.4.1 Adaptive Gradient Clipping . . . 34

4.4.2 Adaptive Noise Scaling . . . 34

4.4.3 Local Training with Opacus . . . 35

4.5 Privacy Accounting . . . 35

4.5.1 Baseline Adap DP-FL (Fu et al. 2022) . . . 35

4.5.2 Hybrid Adap DP-FL + SMPC . . . 36

4.6 SMPC Implementation . . . 37

4.6.1 Secret Sharing Generation . . . 38

4.6.2 Secure Aggregation . . . 38

4.6.3 Reconstruction via Lagrange Interpolation . . . 39

4.7 Checkpointing and Logging . . . 39

4.7.1 Saving Training Progress . . . 39

4.7.2 Logging Configuration . . . 40

4.7.3 CSV Output for Metric Analysis . . . 41

5 Results and Analysis **42**

5.1 Evaluation Metrics . . . 42

5.1.1 Model Utility: Test Accuracy and Test Loss . . . 42

5.1.2 Privacy Spend: Cumulative ε . . . 42

5.2 Results . . . 43

5.2.1 Baseline Adaptive DP-FL Results . . . 43

5.2.2 Hybrid Adap DP-FL + SMPC Results . . . 43

5.3 Analysis . . . 44

5.3.1 Comparison & Trade-Offs . . . 44

6 Conclusion **47**

6.1 Effectiveness of Evaluation Metrics . . . 47

6.2 Conclusion about Research Questions . . . 47

6.3 Limitations . . . 48

6.4 Future Directions . . . 48

References **49**

List of Figures

Figure 1.1 Basic Process of Federated Learning 2

Figure 3.1 End-to-end training pipeline for the hybrid Adap DP-FL + SMPC framework. 18

Figure 5.1 Test accuracy as a function of the global privacy budget ε for two noise settings on MNIST and Fashion-MNIST. 45

Figure 5.2 Test loss as a function of the global privacy budget ε for two noise settings on MNIST and Fashion-MNIST. 46

Acronyms

Adap DP-FL Adaptive Differentially Private Federated Learning

CL-DP Client-Level Differential Privacy

CNN Convolutional Neural Network

CSV Comma-Separated Values

DP Differential Privacy

DP-FL Differentially Private Federated Learning

FL Federated Learning

GPU Graphics Processing Unit

IID Independent and Identically Distributed

Non-IID Non-Independent and Identically Distributed

RDP Rényi Differential Privacy

SGD Stochastic Gradient Descent

SGM Subsampled Gaussian Mechanism

SMPC Secure Multi-Party Computation

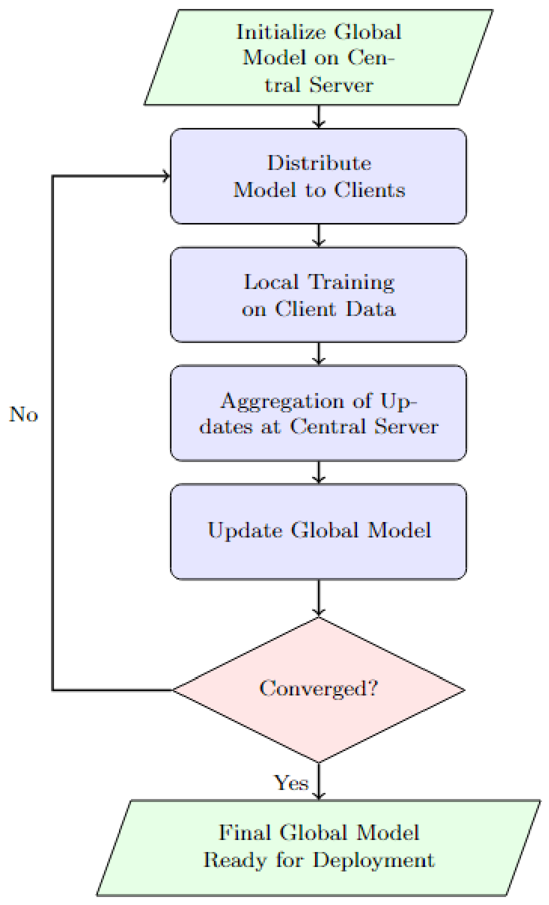
1 Introduction

1.1 Background to the Research

Federated Learning (FL) is a decentralized approach to machine learning that enables multiple parties to collaboratively train a model while keeping their data stored locally (McMahan, Moore, Ramage, Hampson & Arcas (2017*a*)). This methodology is particularly valuable in scenarios where data privacy and security are paramount, such as in healthcare and finance sectors (Sheller et al. (2020)). Unlike traditional centralized machine learning, where data is aggregated into a central repository for training, FL distributes the learning process across multiple devices or nodes. Each participant in the FL process trains a model locally on their dataset and then shares only the model updates (e.g., gradients) with a central server. This server aggregates the updates to improve the global model, which is then redistributed to participants for further training.

Google first introduced the idea of federated learning in 2016 (McMahan, Moore, Ramage, Hampson & Arcas (2017*a*)), primarily to allow Android smartphone users to update models locally without disclosing sensitive personal information. Federated learning was initially applied, to enhance the quality of the language model and safeguard users' private information on the Google Keyboard. After then, Google implemented an application-oriented FL system. Federated learning relies on parameter sharing instead than collecting raw user input, which is the fundamental idea that makes distributed learning possible. Federated Learning offers sensitive data privacy by storing data on users' devices. Because of this unique feature that sets it different from conventional deep learning methods, it has attracted a lot of attention from a variety of applications since it can protect user privacy while promoting collaborative learning. According to the paper Communication-Efficient Learning of Deep Networks from Decentralized Data McMahan, Moore, Ramage, Hampson & Arcas (2017*a*), state their approach "We term our approach Federated Learning, since the learning task is solved by a loose federation of participating devices (which we refer to as clients) which are coordinated by a central server". As the name implies, the model is trained by clients which are in a decentralized location. They download the initial model from a central server and train it on their own private data. Only model updates are sent back to the central server, where they are aggregated and averaged with those from other users to improve the shared model.

Since its inception, FL has undergone significant advancements, driven by the need to



Initialize Global Model on Central Server
Distribute Model to Clients
Local Training on Client Data
Aggregation of Updates at Central Server
Update Global Model
Converged?
No
Yes
Final Global Model Ready for Deployment

Figure 1.1: Basic Process of Federated Learning

enhance data privacy and security while maintaining model performance. Early developments in FL focused on basic protocols for secure model aggregation and addressing data heterogeneity (Yang et al. (2019)). Over time, the integration of privacy-preserving techniques such as Differential Privacy (DP) and Secure Multi-Party Computation (SMPC) has become a central theme in the evolution of FL (Dwork & Roth (2013)).

While FL inherently protects data by keeping it decentralized, the model updates shared in training can still leak sensitive information if not properly protected, as demonstrated in studies on differential privacy and multi-party computation (Dwork & Roth (2013)). Model updates, even without raw data, are vulnerable to privacy attacks such as membership inference, model inversion, and property inference attacks, which can extract sensitive information about the training data (Shokri & Shmatikov (2015)). The risk of such attacks is especially critical in privacy-sensitive fields like healthcare, where patient data must remain confidential. For example, in a healthcare FL system, even though patient data stays within each institution, shared gradients could inadvertently reveal sensitive patient information if the updates are not adequately protected.

The primary challenge in FL is finding an effective balance between privacy and performance. Current privacy-preserving techniques such as Differential Privacy (DP) and Secure Multi-Party Computation (SMPC) address this issue but introduce their own limitations. DP, which injects noise into model updates to obscure specific data points, can decrease model accuracy, while SMPC, which protects data by enabling computations without revealing raw inputs, can significantly increase computational overhead (Geyer et al. (2017*b*)).

This research proposes a novel solution integrating Adaptive Differential Privacy (ADP) with SMPC to address these challenges. The overarching aim of this research is to create a federated learning framework that achieves optimal privacy-performance balance through adaptive DP and SMPC. By advancing the field of FL with these improvements, this study aspires to broaden FL's applicability in privacy-sensitive domains, making collaborative machine learning more secure and effective.

1.2 Problem Statement

Federated Learning (FL) allows many devices to jointly train a machine-learning model without sharing their raw data, but it still risks leaking private information through the gradients that clients send. Differentially Private FL (DP-FL) adds noise to those gradients to protect privacy, yet this often comes at the cost of lower model accuracy and rapid consumption of the privacy budget. The recent Adap DP-FL approach—using adaptive clipping and gradually decreasing noise—improves accuracy, but it still relies on each client injecting noise locally. This local noise both limits the utility of individual updates and requires trusting the server to correctly aggregate noisy gradients.

At the same time, Secure Multi-Party Computation (SMPC) techniques, such as Shamir's secret sharing, can ensure that the server only ever sees encrypted gradient shares, but SMPC has not been combined with adaptive DP-FL. As a result, current DP-FL methods either sacrifice too much accuracy or depend on a semi-trusted server, and they do not take full advantage of SMPC's strong confidentiality guarantees.

Thus, there is a clear gap: no existing solution integrates adaptive differential privacy with SMPC to shift noise injection to the secure aggregation step. Addressing this gap is essential to achieve a better balance between privacy protection (low (ε, δ)) and model performance in real-world federated learning applications.

1.3 Research aim, questions and objectives

1.3.1 Research aim

The primary objective of this research is to design, implement, and evaluate a federated learning framework that seamlessly integrates adaptive differential privacy with secure multi-party computation. By accomplishing these goals, this work seeks to demonstrate a practical pathway to stronger privacy guarantees without sacrificing the utility of federated models in real-world distributed learning scenarios.

1.3.2 Research questions

1. **How can Secure Multi-Party Computation (SMPC) be used to improve privacy in Adaptive Differentially Private Federated Learning (Adap DP-FL)?**

   This research question explores how Secure Multi-Party Computation (SMPC) can be integrated into Adaptive Differentially Private Federated Learning (Adap DP-FL) to better protect client privacy. Instead of sending their gradients directly, each client securely splits and shares their adaptively clipped updates. The central server then only sees the combined result—not any individual client's data—before applying differential privacy noise. Through experiments on widely used federated learning benchmarks, we will demonstrate that this SMPC-based aggregation preserves model utility while significantly reducing the risk of per-client data exposure and minimizing the trust placed in the server.

2. **How can we theoretically prove that the proposed hybrid approach satisfies the required differential privacy guarantees?**

   This objective is to provide a rigorous mathematical analysis showing that our hybrid DP-SMPC mechanism computes its privacy loss correctly, following the formal rules of differential privacy. By carrying out this theoretical validation, we will prove that our privacy accountant accurately tracks the global (ε, δ) budget. We will then use these analytical results to guide and interpret our empirical evaluations.

1.3.3 Research objectives

- Develop a federated learning system that integrates SMPC—via Shamir's secret sharing—with adaptive differential privacy techniques (adaptive gradient clipping and noise-scaling).
- Analyze how SMPC protects individual client updates during aggregation and reduces the trust assumptions placed on the central server.
- Provide a formal proof that the hybrid DP-SMPC scheme achieves the target privacy parameters (ε, δ)
- Compare model accuracy between the SMPC-enabled DP-FL system and a baseline Adap DP-FL system without SMPC, under the same overall privacy budget.

1.4 Significance of the Project

From a computer science point of view, this project pushes the boundaries of privacy in federated learning by combining two powerful ideas—adaptive differential privacy and secure multi-party computation—into one practical system. Instead of each client adding noise individually, we shift the noise addition to the final secure aggregation step using Shamir's secret sharing. This reduces the amount of privacy budget each client needs to spend and makes our overall privacy analysis tighter, all while keeping model accuracy high, even under strict privacy limits. The prototype we've built, along with a formal privacy tracker, can act as a helpful guide for future researchers working on privacy-preserving machine learning.

In real-world terms, our method makes federated learning safer and more trustworthy by reducing how much we need to rely on central servers or secure communication networks. This is especially useful for sensitive areas like healthcare, finance, or IoT systems, where keeping personal data private is really important. Our approach helps organizations meet privacy laws like GDPR and HIPAA, while still building models that actually work well. By showing that strong privacy doesn't have to mean poor performance, this research opens the door for wider use of federated learning in everyday, privacy-sensitive applications.

1.5 Research Approach and Methodology

This research adopts a structured and iterative methodology to design, implement, and evaluate a privacy-preserving federated learning system that integrates Adaptive Differential Privacy (Adap DP-FL) with Secure Multi-Party Computation (SMPC). The methodology comprises the following key stages:

1. **Recreation of Baseline Models**

   To establish a benchmark for performance and privacy comparison, the first step involves recreating relevant baseline model:

   - An Adaptive Differentially Private Federated Learning (Adap DP-FL) model, implemented based on the approach proposed by Fu et al. (2022) incorporating adaptive clipping and noise scaling.

   This baseline model serve as references to evaluate the effectiveness of the proposed hybrid method.

2. **Architectural Design of the Hybrid DP-SMPC Model**

   Following the baselines, the research focuses on designing the architecture of the proposed hybrid approach that combines adaptive differential privacy with secure aggregation through SMPC. This involves:

   - Defining the secure gradient sharing mechanism (e.g., Shamir's secret sharing).

   - Identifying optimal points for noise injection.
   - Ensuring compatibility with existing adaptive clipping and privacy accounting techniques.

3. **Theoretical Validation of Privacy Guarantees**

   A critical part of the methodology involves analyzing and validating the privacy accounting of the hybrid model. This includes:

   - Reviewing literature and mathematical frameworks for differential privacy in secure aggregation settings.
   - Demonstrating that the hybrid model satisfies differential privacy guarantees across multiple training rounds.
   - Ensuring noise calibration and sensitivity analysis are theoretically sound.

4. **Implementation of the Proposed Hybrid Model**

   The designed hybrid architecture is implemented in a simulated federated learning environment. Core components include:

   - Local training with adaptive clipping.
   - Secret-sharing of gradients.
   - Secure aggregation at the server followed by global noise addition.
   - Differential privacy tracking using composition theorems.

5. **Experimental Evaluation and Benchmarking**

   The final stage involves experimental validation of the proposed system using benchmark datasets such as MNIST and Fashion-MNIST.The results are compared across all models to assess the effectiveness of the hybrid approach in balancing privacy and performance in federated learning.

1.6 Outline of the Dissertation

This dissertation is organized into six chapters, each building upon the previous to provide a comprehensive investigation into privacy-preserving federated learning using a hybrid approach that integrates Adaptive Differential Privacy and Secure Multi-Party Computation.

- **Chapter 1: Introduction** provides the background and motivation for the study, defines the problem statement, outlines the research questions, objectives, and significance of the project, and briefly introduces the methodology and scope of the work.

- **Chapter 2: Background and Literature Review** offers foundational knowledge on federated learning and its key challenges, with a particular focus on privacy threats and preservation mechanisms. It introduces core concepts such as Differential Privacy and Secure Multi-Party Computation (SMPC), surveys existing methods that balance privacy and utility, and identifies the research gap that this thesis aims to address.
- **Chapter 3: Design** details the high-level architectural design of the proposed system. It contrasts the baseline Adaptive DP-FL framework with the proposed hybrid approach, illustrating component interactions, data flows, and the overall privacy accounting mechanisms involved.
- **Chapter 4: Implementation** documents the technical setup and development workflow. It describes the datasets, model architectures, local training process using Opacus, and key privacy-preserving components including adaptive gradient clipping, adaptive noise scaling, SMPC-based secure aggregation using Shamir's Secret Sharing, and logging and checkpointing mechanisms.
- **Chapter 5: Results and Analysis** presents the experimental outcomes on MNIST and Fashion-MNIST datasets. It evaluates both the baseline and hybrid methods using metrics such as test accuracy, test loss, and global privacy budget (ε). It also provides a comparative analysis to highlight trade-offs between utility and privacy under varying noise settings.
- **Chapter 6: Conclusion** summarizes the major findings, answers the research questions based on empirical evidence, discusses the limitations of the study, and proposes several directions for future research in privacy-preserving federated learning.

1.7 Delimitation of Scope

1.7.1 In Scope

The following aspects will be covered under the research project.

- Development of a federated learning setup using standard datasets (e.g., MNIST, Fashion-MNIST) in a simulated environment.
- Design and implementation of the hybrid approach, integrating SMPC with Adap DP-FL.
- Privacy accounting and theoretical validation of differential privacy guarantees in the hybrid system.
- Empirical evaluation of model utility and privacy loss (ε, δ)

1.7.2 Out Of Scope

The following aspects will be out of the project scope

- Real-world deployment on mobile devices or production federated learning platforms
- Comparison with other advanced privacy-preserving techniques, such as homomorphic encryption etc.
- Formal cryptographic security proofs of the SMPC protocol (focus is on practical implementation and privacy gain).

2 Background & Literature review

Federated learning is a machine learning setting which encompasses multiple clients working together to solve a machine learning problem, with a central server or service provider coordinating the effort where each clients' raw data is stored locally, with model updates are transferred for immediate aggregation to achieve the learning objectives.

2.1 Training Process

The following steps will explain the model training process of a typical federated learning system which encompasses the FedAverage algorithm of McMahan, Moore, Ramage, Hampson & Arcas (2017*a*). There can be many variations of this, but this can be considered as a common template. The training procedure will be coordinated by a central server, which will keep repeating the same steps until the training is stopped (Liu et al. (2022)).

1. Client selection - The server takes samples from a group of clients.
2. Broadcast - The chosen clients download a training program along with the current model weights.
3. Client computation - Using the training program, which may, for example, run Stochastic gradient descent on the local data (as in Federated Averaging), each chosen device locally computes an update to the model.
4. Aggregation - An aggregate of the device updates is gathered by the server. Additionally, this process makes use of a variety of techniques, including secure aggregation, noise addition, to maintain privacy, and techniques to improve communication efficiency.
5. Model update - The shared model is updated locally by the server using the aggregated updates that was calculated from the clients that took part in the current round.

2.2 Key Challenges

Federated learning (FL) must overcome several interconnected challenges to be practical and effective in real-world deployments:

- **Statistical Heterogeneity.** In FL, each client's local data often follow different distributions (so-called non-IIDness), which can slow convergence and degrade the accuracy of the global model. Standard aggregation methods such as FedAvg assume homogeneous data and may perform poorly under high heterogeneity (Zhao et al. 2018, McMahan, Moore, Ramage, Hampson & y Arcas 2017).

- **Communication Overhead.** FL requires frequent transmission of model parameters between the central server and potentially thousands of edge devices. Limited bandwidth, high latency, or intermittent connectivity make such synchronization costly. Techniques like increasing local computation (multiple epochs) (McMahan, Moore, Ramage, Hampson & y Arcas 2017), structured updates (Konečný et al. 2016), and client subsampling are commonly used to reduce communication rounds.

- **Privacy Preservation.** Although raw data never leave client devices, exchanged model updates can still leak sensitive information via model-inversion or membership-inference attacks (Melis et al. 2019, Shokri et al. 2017). Consequently, FL must integrate formal privacy measures such as differential privacy (Abadi et al. 2016) and cryptographic protocols like secure aggregation (Bonawitz et al. 2017) to provide provable confidentiality.

Together, these challenges—statistical heterogeneity, communication constraints, and rigorous privacy requirements—define the core research questions addressed throughout this thesis.

2.2.1 Privacy Challenges in Federated Learning

Although federated learning (FL) never shares raw data, recent studies have demonstrated that exchanging model updates still exposes sensitive information (Melis et al. 2019, Shokri et al. 2017). In particular, most privacy breaches occur during the *inference phase*, when the collaboratively trained model is deployed to make predictions on new inputs (Rafi et al. 2024). Adversaries exploit released model parameters or queried outputs to mount a variety of attacks:

1. **Membership Inference Attacks** (Shokri et al. 2017): Determine whether a specific record was part of a client's training set by observing the model's confidence or loss on that example.

2. **Model Inversion Attacks** (Fredrikson et al. 2015): Reconstruct approximate input features (e.g. images or attributes) from the model's outputs or gradients, effectively "inverting" the model to reveal private training data.

3. **Property Inference Attacks** (Ateniese et al. 2015): Infer global or client-specific attributes (e.g. demographic statistics) that are correlated with the training data but not intended for disclosure.

4. **Model Poisoning Attacks** (Bagdasaryan et al. 2020): Malicious clients craft and submit carefully designed updates to introduce backdoors or degrade the overall performance of the global model.

5. **Data Poisoning Attacks** (Muñoz-González et al. 2017): Adversaries contaminate the training data—either by injection or modification—to bias the learned model or cause it to fail on particular inputs.

Addressing these threats requires integrating rigorous privacy-preserving techniques (e.g. differential privacy (Abadi et al. 2016), secure aggregation (Bonawitz et al. 2017)) as well as robust anomaly detection to guard against malicious participants.

2.3 Privacy Preservation in Federated Learning

The federated framework presents unique challenges for current privacy-preserving algorithms. It is critical to develop strategies that are not only efficient in computation and communication, but also adaptable to decentralized participants, all while maintaining high model accuracy. To address the above-mentioned challenges and privacy attacks federated learning has utilized following privacy preserving mechanisms.

2.3.1 Differential Privacy

Differential Privacy (DP) is a rigorous mathematical framework for protecting individual data points in statistical computations. It was introduced by Dwork et al. (Dwork et al. 2006) and has since become one of the most widely used techniques for privacy preservation in machine learning.

What is Differential Privacy?

Intuitively, a mechanism is differentially private if its output does not change significantly when a single individual's data is added or removed. This ensures that no attacker can confidently determine whether a particular person's data was used in the training process—thereby preserving privacy.

Formally, a randomized algorithm $\mathscr{M}$ is said to satisfy (ε, δ)-differential privacy if for any two datasets D and D' that differ by a single record, and for any possible output S,

$$\Pr[\mathscr{M}(D) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathscr{M}(D') \in S] + \delta.$$

Here:

- ε (epsilon) measures the privacy loss—a smaller ε implies stronger privacy.
- δ is a small probability that the guarantee may not hold.

This definition ensures that individual data points have minimal influence on the final output of the model (Dwork et al. 2006, Abadi et al. 2016).

Use of Differential Privacy in Federated Learning

In Federated Learning (FL), model training is distributed across clients, and only model updates are shared. However, even model updates can leak private information through various inference attacks (Zhu et al. 2019, Melis et al. 2019). DP helps mitigate this risk by introducing controlled noise during training.

There are two main ways to apply differential privacy in FL:

1. **Client-level DP (Local Noise):** Each client clips its model update and adds noise before sending it to the server. This ensures that the update alone does not leak information about the client's data.

2. **Server-level DP (Global Noise):** The server aggregates all client updates and adds noise to the aggregated result. This can provide the same privacy guarantee with less overall noise if secure aggregation (e.g., SMPC) is used (McMahan, Moore, Ramage, Hampson & Arcas 2017*b*, Truex et al. 2019).

Limitations of Differential Privacy

Despite its strong theoretical guarantees, DP comes with practical challenges:

- **Utility–Privacy Tradeoff:** Adding noise to ensure privacy often reduces model accuracy. Striking the right balance between utility and privacy requires careful tuning.

- **Complex Accounting:** In real-world training scenarios (e.g., many epochs, non-IID data), tracking the cumulative privacy loss over time can be challenging. Frameworks like Rényi Differential Privacy (RDP) (Mironov 2017) help address this.

- **Noise Sensitivity:** Models trained on small datasets or with high learning variance may be more sensitive to DP noise, resulting in unstable convergence.

Despite these limitations, differential privacy remains a cornerstone of privacy-preserving federated learning due to its formal guarantees and general applicability.

2.3.2 Secure Multi-Party Computation (SMPC) and Secret Sharing

Secure Multi-Party Computation (SMPC) is a class of cryptographic protocols that allows multiple parties to jointly compute a function over their inputs while keeping those inputs private (Yao 1982). This means that no individual party learns anything about the others' data beyond what can be inferred from the final output.

What is Secret Sharing?

Secret sharing is a foundational building block of SMPC. It allows a secret (e.g., a private value) to be split into multiple parts, called *shares*, and distributed to different parties. No single share reveals any information about the secret. The original secret can only be reconstructed when a certain number of shares are combined. This enables collaborative computations where privacy is preserved unless a threshold number of parties collude.

Types of Secret Sharing

There are two primary types of secret sharing schemes:

- **Additive Secret Sharing:** The secret is split by generating random values that sum up to the original value. For example, to share s, generate s_1, s_2 randomly such that $s_3 = s - (s_1 + s_2)$, and distribute (s_1, s_2, s_3) to three parties.
- **Shamir's Secret Sharing (SSS):** Introduced by Shamir (Shamir 1979), this scheme uses polynomial interpolation. The secret is the constant term of a random polynomial, and each share is a point on that polynomial. Only a threshold number of shares are needed to reconstruct the original value using Lagrange interpolation.

How Shamir's Secret Sharing Works

In Shamir's Secret Sharing, to share a secret s, a random polynomial $f(x) = s + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$ of degree $t-1$ is constructed. Then:

- Each participant receives a share $f(i)$, where i is the participant's identifier.
- Any group of t or more participants can reconstruct s using polynomial interpolation (e.g., Lagrange interpolation).
- Fewer than t participants learn nothing about the secret, ensuring perfect privacy.

This method is fault-tolerant and collusion-resistant, making it well-suited for secure aggregation in federated learning.

Use of SMPC in Federated Learning

In Federated Learning (FL), SMPC helps protect model updates from individual clients. Instead of sending raw gradients, each client secret-shares its update and sends encrypted shares to multiple servers. The server-side aggregation is then performed on these shares:

- **Input Secrecy:** No single server has access to the original update.
- **Robust Aggregation:** Aggregated model updates can be reconstructed only if a threshold number of shares are available.

- **Privacy Boost:** When combined with Differential Privacy, SMPC ensures that sensitive information is protected both during transmission and after aggregation (Bonawitz et al. 2017, Truex et al. 2019).

This enables FL systems to inject DP noise *after* aggregation, reducing the amount of noise required and preserving model utility.

Limitations of SMPC

While SMPC provides strong privacy guarantees, it has some challenges:

- **Increased Communication:** Clients must send multiple shares (e.g., one per server), increasing bandwidth usage.
- **Server Trust Model:** Correct reconstruction assumes that not all aggregation servers are colluding.
- **Computational Overhead:** Interpolation and share aggregation introduce extra computation.
- **Scalability:** SMPC protocols may become complex in large-scale deployments with many clients and servers.

Despite these limitations, SMPC is a powerful method to strengthen privacy in federated settings when used in conjunction with other mechanisms like DP.

2.4 Balancing Privacy and Utility in Federated Learning

In Federated Learning (FL), there is an inherent trade-off between maintaining user privacy and ensuring high model utility. Various studies have aimed to strike this balance through mechanisms such as Differential Privacy (DP), Secure Multi-Party Computation (SMPC), homomorphic encryption, and hybrid techniques. However, each approach presents its own set of limitations in either privacy protection, model accuracy, or system scalability.

Differential Privacy-Based Approaches

The most commonly adopted method for privacy preservation in FL is client-side DP, where noise is added directly to each client's model update before aggregation (Geyer et al. 2017*a*, McMahan et al. 2018). This ensures that each client's contribution remains indistinguishable. However, the local application of noise increases overall variance and often results in substantial accuracy degradation—especially in high-dimensional models or when the number of clients is small.

SMPC and Cryptographic Aggregation

Another set of approaches relies on cryptographic techniques like SMPC and homomorphic encryption to perform secure aggregation (Bonawitz et al. 2017). These methods protect updates during transmission but do not inherently guarantee differential privacy. Therefore, they must often be combined with DP for full privacy guarantees. While SMPC provides strong input secrecy, it adds communication and computation overhead, and usually requires coordination among a fixed number of trusted aggregation servers.

Hybrid Methods and Adaptive Mechanisms

To improve the privacy–utility trade-off, several recent works propose hybrid methods. These combine DP and SMPC by delaying the addition of noise until after secure aggregation (Truex et al. 2019). This reduces the amount of noise needed for privacy guarantees, as the sensitivity of the aggregated update is much lower than individual updates. However, most hybrid systems still use fixed clipping thresholds and static noise levels, which are suboptimal as training progresses.

Adaptive DP-FL Framework

One notable solution that addresses these limitations is the Adaptive Differentially Private Federated Learning (Adap DP-FL) framework (Fu et al. 2022). This method introduces:

- **Adaptive Clipping:** Each client's gradient clipping threshold is adjusted dynamically based on the previous round's average gradient norm.
- **Noise Decay:** The Gaussian noise multiplier is gradually decreased when validation loss improves, allowing the model to fine-tune more precisely in later rounds.

While Adap DP-FL significantly improves model utility under tight privacy budgets, it still applies noise locally at each client. This increases overall noise variance and leads to unnecessary degradation in utility, especially when many clients are involved. Additionally, since clipping and noise are client-specific, the system must track privacy budgets individually, making coordination more complex.

2.5 Research Gap

Despite these advancements, there remains a need for a framework that:

- Dynamically adapts both clipping and noise in response to training feedback,
- Avoids per-client noise variance by using centralized noise addition,
- Preserves privacy with provable guarantees using RDP analysis,
- Maintains input secrecy using secure aggregation protocols.

This research addresses this gap by proposing a hybrid Adap DP-FL + SMPC framework. The system combines adaptive clipping and adaptive noise decay mechanisms with post-aggregation Gaussian perturbation enabled by Shamir's secret sharing. By securely aggregating updates and adding a single global noise term at the server, the approach reduces noise variance, simplifies privacy tracking, and enhances model performance while respecting strict privacy budgets.

3 Design

3.1 High-Level Framework Overview

This chapter presents the design of our hybrid federated learning framework that combines Adaptive Differential Privacy (Adap DP) with Secret-Sharing-based Secure Multi-Party Computation (SMPC). The goal is to retain the strong utility of the baseline Adap DP-FL method while further reducing trust in the central server by never revealing individual client updates in the clear. In each training round, clients locally compute and adaptively clip their gradients, then secret-share those clipped gradients across multiple aggregation servers. Only the aggregate gradient is ever reconstructed, to which differential privacy noise is then added before updating the global model. Validation-driven noise scaling ensures we maintain a target (ε, δ) budget while preserving accuracy.

3.1.1 Interaction of Components

The three main components—Federated Learning, Adaptive Differential Privacy, and SMPC—interlock as follows:

- **Federated Learning (FL):**
  - Orchestrates rounds of local training on each client.
  - Aggregates client updates to produce a new global model.
- **Adaptive Differential Privacy (Adap DP):**
  - Applies per-client, per-round clipping thresholds C_k^t based on previous gradient norms.
  - Injects decreasing Gaussian noise $\mathcal{N}(0, (\sigma_t \cdot C)^2)$ into the aggregated gradient, with σ_t decayed when validation loss has fallen for several rounds.
  - Keeps a full RDP accounting to ensure the desired (ε, δ) over all T rounds.
- **Secure Multi-Party Computation (SMPC):**
  - Each client secret-shares its clipped gradient across three or more non-colluding servers.
  - Servers sum their shares locally; only the combined sum is ever reconstructed.

– No individual server ever sees a raw client update, drastically reducing trust requirements.

Together, these produce a pipeline in which no raw gradient leaves a client, yet a differentially private and accurate global model is still learned.

3.1.2 Pipeline Flowchart

Figure 3.1 shows the end-to-end flow of a single training round in our hybrid system:

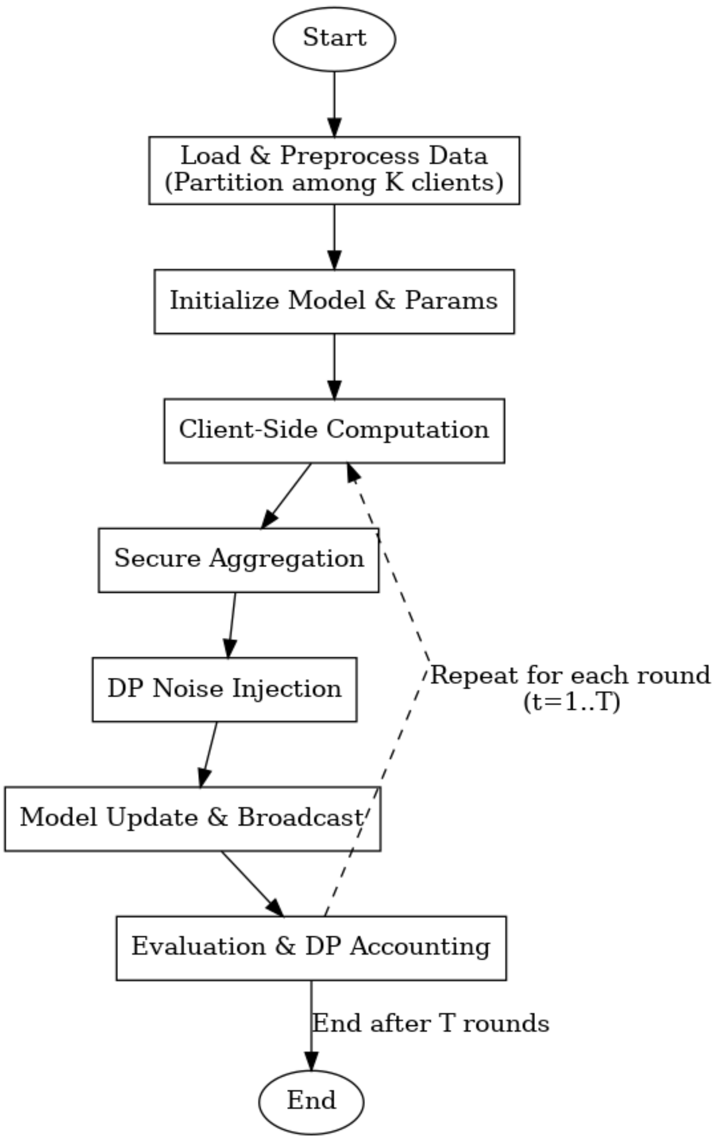

Start
Load & Preprocess Data (Partition among K clients)
Initialize Model & Params
Client-Side Computation
Secure Aggregation
DP Noise Injection
Model Update & Broadcast
Evaluation & DP Accounting
Repeat for each round (t=1..T)
End after T rounds
End

Figure 3.1: End-to-end training pipeline for the hybrid Adap DP-FL + SMPC framework.

This pipeline ensures that:

1. Clients never expose raw gradients.
2. Aggregation servers only handle secret shares.
3. Differential privacy noise is injected exactly once, on the global sum.

4. Full privacy accounting via RDP tracks cumulative ε.

3.2 Baseline System Architecture

Federated Learning (FL) enables decentralized training across multiple client devices, each holding local data, without transferring raw data to a central server. However, this decentralized setup still poses privacy risks since model updates can unintentionally leak information about individual data points (Melis et al. 2019, Nasr et al. 2019). To mitigate such threats, Differential Privacy (DP) is often employed to ensure that the inclusion or exclusion of a single data point has a limited effect on the output (Dwork et al. 2006).

A notable advancement in this area is the Adaptive Differentially Private Federated Learning (Adap DP-FL) framework proposed by Fu et al. (Fu et al. 2022). This approach enhances standard DP-FL by introducing two key techniques:

(a) **Adaptive Gradient Clipping**, where each client's clipping threshold is adjusted dynamically using differentially private estimates of the previous gradient norms.

(b) **Adaptive Noise Scaling**, where the amount of noise added to each client's update is proportional to its clipping threshold, ensuring better utility without sacrificing privacy.

In Adap DP-FL, each client clips its gradients based on a noisy version of the previous round's average gradient norm and adds calibrated Gaussian noise to satisfy (ε, δ)-DP. The central server performs weighted aggregation of these noised gradients to update the global model. Privacy loss is tracked across training rounds using Rényi Differential Privacy (RDP) accounting (Mironov 2017), which allows tighter analysis compared to traditional composition theorems.

By adapting both the clipping threshold and noise scale, Adap DP-FL achieves a better balance between model performance and privacy, especially under heterogeneous data distributions and non-IID client scenarios (Fu et al. 2022).

3.2.1 Workflow

In Adaptive Differentially Private Federated Learning (Adap DP-FL), each client performs local training and ensures that its model updates satisfy differential privacy before sending them to the central server. The client-side mechanism incorporates adaptive clipping and adaptive noise addition, both of which are designed to balance privacy and model utility.

1. **Step 1: Local Gradient Computation**

   Each client k computes its gradient g_k^t at round t based on its local dataset D_k and the current global model parameters θ^t:

$$g_k^t = \nabla \mathscr{L}(\theta^t; D_k) \tag{3.1}$$

where $\mathscr{L}$ denotes the local loss function (e.g., cross-entropy), and $\nabla\mathscr{L}$ is the gradient with respect to the global model (McMahan, Moore, Ramage, Hampson & Arcas 2017*a*).

2. **Step 2: Gradient Clipping with Adaptive Threshold**

   To prevent large gradients from dominating the training and to bound sensitivity for differential privacy, each client clips its gradient based on a clipping norm C_k^t, which is computed adaptively:

   $$\bar{g}_k^t = \frac{g_k^t}{\max\left(1, \frac{\|g_k^t\|_2}{C_k^t}\right)} \tag{3.2}$$

   Here, $\bar{g}_k^t$ is the clipped gradient, $\|g_k^t\|_2$ is the L2 norm of the gradient, and C_k^t is the clipping threshold that changes over time depending on previous gradients (Andrew et al. 2019).

3. **Step 3: Adaptive Estimation of Clipping Threshold**

   The clipping threshold C_k^t is estimated based on the previous round's gradient norms with Gaussian noise added for privacy:

   $$\tilde{C}_k^t = \alpha \cdot \hat{n}_k^{t-1} + \mathcal{N}(0, \sigma_c^2) \tag{3.3}$$

   where:

   - $\hat{n}_k^{t-1}$ is the average L2 norm of previous clipped gradients.
   - α is a scaling factor.
   - σ_c is the noise standard deviation added for privacy (Fu et al. 2022).

4. **Step 4: Adding Gaussian Noise (Per-Client)**

   After clipping, Gaussian noise is added to the gradient to ensure differential privacy:

   $$\tilde{g}_k^t = \bar{g}_k^t + \mathcal{N}(0, \sigma_k^2 C_k^2) \tag{3.4}$$

   where:

   - σ_k is the client-specific noise multiplier.
   - C_k is the clipping threshold.

   The amount of noise scales proportionally with the clipping threshold, allowing clients with larger updates to contribute more, while still maintaining privacy (Abadi et al. 2016).

5. **Step 5: Gradient Descent Update**

$$\theta_k^{t+1} = \theta_k^t - \eta \tilde{g}_k^t \tag{3.5}$$

   where:

   - θ_k^t is the local model of client k at round t,
   - η is the learning rate,
   - $\tilde{g}_k^t$ is the differentially private gradient after clipping and noise addition.

   This step updates the client's local model before potential aggregation at the server.

6. **Step 6: Privacy Budget Computation**

   Each client maintains a local privacy budget, which accumulates over rounds. Using Rényi Differential Privacy (RDP), the cumulative privacy loss ε_k^t at round t is estimated:

$$\varepsilon_k^t = \mathrm{RDP2DP}(\textstyle\sum_{\tau=1}^{t} \mathrm{RDP}_\tau, \delta) \tag{3.6}$$

   This step uses standard composition theorems under the RDP framework to track how much privacy has been spent over time (Mironov 2017, Wang et al. 2019).

7. **Step 7: Privacy Budget Check and Early Termination**

   To prevent over-spending the client's privacy budget, the system checks if the privacy loss has exceeded a threshold:

$$\text{if } \varepsilon_k^t > \varepsilon_{\max}, \quad \text{then stop participation} \tag{3.7}$$

   where $\varepsilon_{\max}$ is the maximum allowable privacy loss per client (Abadi et al. 2016).

8. **Step 8: Upload Noised Update to Server**

   If the privacy budget has not been exhausted, the client uploads its noised update $\tilde{g}_k^t$ (or the updated model parameters θ_k^{t+1}, depending on protocol) to the server for aggregation:

$$\text{Client } k \longrightarrow \text{Server}: \quad \tilde{g}_k^t \text{ or } \theta_k^{t+1} \tag{3.8}$$

   This ensures that only privacy-preserving updates are shared, reducing the risk of information leakage while enabling collaborative model training.

9. **Step 9: Server-side Aggregation**

   Once the clients have uploaded their noised and clipped gradients $\tilde{g}_k^t$, the server performs weighted model aggregation. This step combines the client updates to form the next global model:

$$w^{t+1} = \sum_{k \in \mathcal{K}} p_k \cdot w_k^{t+1} \tag{3.9}$$

where:

- w^{t+1} is the updated global model,
- w_k^{t+1} is the model from client k,
- $p_k = \frac{|D_k|}{\sum_{j \in \mathcal{K}} |D_j|}$ is the aggregation weight based on the relative size of client k's dataset,
- $\mathcal{K}$ is the set of participating clients in the current round.

This federated averaging step ensures that clients with more data have a proportionally larger influence on the updated model (McMahan, Moore, Ramage, Hampson & Arcas 2017*a*).

10. **Step 10: Adaptive Noise Control**

    To enhance utility without compromising privacy, the server monitors model performance trends. If the validation loss does not improve over several rounds, the global noise multiplier σ_t may be decayed using a decay factor β:

    $$\text{if } J(w^{t-2}) > J(w^{t-1}) > J(w^t) \quad \Rightarrow \quad \sigma_{t+1} = \beta \cdot \sigma_t \tag{3.10}$$

    where:

    - $J(w^t)$ is the validation loss for the global model after round t,
    - $\beta \in (0,1)$ is the noise decay factor (Fu et al. 2022).

    This adaptive control helps improve convergence while still maintaining the privacy constraints.

11. **Step 11: Global Update Broadcast**

    After completing the aggregation and (optional) noise adjustment, the server broadcasts the updated global model w^{t+1} and current noise multiplier σ_{t+1} to all clients. These values are then used as the starting point for the next training round:

    $$\text{Server } \rightarrow \text{Clients}: \quad w^{t+1}, \; \sigma_{t+1} \tag{3.11}$$

    This broadcast ensures synchronization across all participating clients and marks the beginning of the next federated round.

3.3 Proposed Hybrid Architecture (Adap DP-FL + SMPC)

3.3.1 Client-Side Process

In the proposed hybrid system, the client is responsible for performing local model training, applying adaptive gradient clipping, and secret-sharing the model updates with the server. Unlike the baseline Adap DP-FL, the client does not add noise locally; instead, secure aggregation and noise addition are handled centrally. The steps performed at each client during a federated round are described below.

1. **Local Model Training**
   Each client downloads the global model w_t and performs one epoch of training using its local dataset D_k. The gradient $g_k^{(i)}$ for a mini-batch sample x_i is computed using:

$$g_k^{(i)} = \nabla \mathscr{L}(w_t; x_i) \tag{3.12}$$

   where $\mathscr{L}$ is the local loss function (e.g., cross-entropy), and w_t is the global model at round t (McMahan, Moore, Ramage, Hampson & Arcas 2017*a*).

2. **Adaptive Gradient Clipping**
   To bound sensitivity for differential privacy, clients clip gradients using a threshold C_k^t that adapts over time:

$$\bar{g}_k^{(i)} = \frac{g_k^{(i)}}{\max\left(1, \frac{\|g_k^{(i)}\|_2}{C_k^t}\right)} \tag{3.13}$$

   where $\|g_k^{(i)}\|_2$ is the L2 norm of the gradient. This ensures all gradients are bounded by C_k^t (Andrew et al. 2019).

3. **Adaptive Clipping Threshold Estimation**
   The threshold C_k^t is updated at each round based on the previous clipped gradient norms. A Gaussian noise term is included to obscure the exact norm:

$$C_k^t = \alpha \cdot \hat{n}_k^{t-1} + \mathcal{N}(0, \sigma_c^2) \tag{3.14}$$

   where:

   - $\hat{n}_k^{t-1}$: average L2 norm of clipped gradients from previous round.
   - α: adaptive scaling factor.
   - σ_c: noise standard deviation for clipping estimation.

   This method ensures that the clipping threshold responds to training dynamics while maintaining privacy (Fu et al. 2022).

4. **Averaging Clipped Gradients**

   After processing the mini-batches, the client computes the average clipped gradient:

$$\bar{g}_k^t = \frac{1}{|L_k^t|} \sum_{i \in L_k^t} \bar{g}_k^{(i)} \tag{3.15}$$

   where L_k^t is the set of sampled data points used for training at round t.

5. **Model Update Computation**

   The local update is computed as the difference between the updated local model and the received global model:

$$\Delta_k^t = w_k^{t+1} - w_t \tag{3.16}$$

   This update Δ_k^t is the quantity that will be secret-shared instead of sent directly.

6. **Secret Sharing of Update**

   The computed local update Δ_k^t is split into three shares using a linear Shamir's secret sharing scheme:

$$f(x) = \Delta_k^t + ax \tag{3.17}$$

   where a is a randomly sampled noise tensor, and shares are:

$$s_1 = f(1), \quad s_2 = f(2), \quad s_3 = f(3)$$

   These shares are sent to three servers (or parties) such that any two of them can reconstruct the original update, while individual shares reveal nothing about the original data (Shamir 1979).

3.3.2 Server-Side Process

The server in the proposed hybrid system plays a crucial role in securely aggregating the client updates, injecting noise for differential privacy, and accounting for the privacy budget. This section outlines each component of the server-side workflow in detail.

1. **Receiving Secret Shares**

   Each client computes the local update Δ_k^t as the difference between the locally updated model and the current global model:

$$\Delta_k^t = \theta_k^t - \theta^t \tag{3.18}$$

   where θ_k^t is the local model at client k, and θ^t is the global model at round t.

   This update is then split into $n = 3$ shares using a degree-1 Shamir secret sharing scheme:

$$f(x) = \Delta_k^t + ax \tag{3.19}$$

   where a is a randomly sampled tensor of the same shape as Δ_k^t. The server receives two of the shares (e.g., $f(1), f(2)$) from each client.

2. **Secure Aggregation of Shares**

   For each model parameter, the server aggregates the corresponding shares across all participating clients:

$$\text{Aggregated Share}_j = \sum_{k=1}^{K} \text{Share}_{k,j} \tag{3.20}$$

   where $j \in \{1,2\}$ represents the selected share index for reconstruction, and K is the number of clients.

3. **Reconstruction of Aggregated Update**

   Using Lagrange interpolation over two aggregated shares, the server reconstructs the aggregate update:

$$\Delta^t = \frac{2}{1} \cdot \text{Aggregated Share}_1 - \frac{1}{1} \cdot \text{Aggregated Share}_2 \tag{3.21}$$

   This ensures that individual client updates are never revealed, only their aggregate.

4. **Adding Global Gaussian Noise**

   To ensure differential privacy, the server adds Gaussian noise to the aggregated update:

$$\tilde{\Delta}^t = \Delta^t + \mathcal{N}(0, \sigma^2 \cdot (C/K)^2) \tag{3.22}$$

   where σ is the global noise multiplier, C is the maximum clipping norm across clients, and K is the number of clients.

5. **Updating the Global Model**

   The noisy aggregated update is then used to update the global model using a learning rate η:

$$\theta^{t+1} = \theta^t + \eta \cdot \tilde{\Delta}^t \tag{3.23}$$

6. **Privacy Accounting Using RDP**

   The server tracks the cumulative privacy loss using the Rényi Differential Privacy (RDP) framework. The training process terminates when ε exceeds the maximum allowed budget.

3.3.3 Privacy Accounting

In our hybrid Adap DP-FL + SMPC system, all private information release occurs through the Gaussian mechanism applied *after* secure aggregation. Because secret-sharing and aggregation are *post-processing* of client data, they do not affect the differential privacy guarantee (post-processing theorem) (Dwork et al. 2006). We therefore account privacy loss solely based on the noise added at the server.

How We Measure Privacy in One Round:

In each training round, the server adds Gaussian noise to the aggregated update. The amount of privacy loss incurred from this step is measured using Rényi Differential Privacy (RDP), which quantifies the information leakage of a randomized mechanism (like adding noise). The privacy cost of a single round depends on:

- The **sampling ratio** $q = \frac{B}{N}$, which is the probability of including each data point (where B is batch size and N is total dataset size).
- The **clipping norm** C, which limits how much a single client's update can influence the output.
- The **noise multiplier** σ, which controls the scale of Gaussian noise.
- The **Rényi order** α, which determines how sensitive the privacy bound is to tail behavior.

How Noise Differs from Adap DP-FL:

Unlike standard Adap DP-FL (Fu et al. 2022), which adds noise individually at the **client side** before uploading, our hybrid framework leverages **Secure Multi-Party Computation (SMPC)** to first aggregate clipped gradients across all clients *without revealing individual updates.* Gaussian noise is then added once to the aggregated result at the server. This design improves utility for two key reasons:

a. Reduced Sensitivity of Aggregated Updates:
In Adap DP-FL, each client adds noise scaled to their local clipping threshold C_k. The global sensitivity—defined as the maximum influence of any single client—is thus bounded by the maximum clipping threshold $C_{\max} = \max_k C_k$.

In our hybrid framework, however, the sensitivity of the aggregated update is reduced to:

$$\Delta = \frac{C_{\max}}{K}$$

where K is the number of clients. This is because SMPC averages the contributions across K clients, inherently scaling each client's influence by a factor of $1/K$.

b. Lower Total Noise Variance:
In Adap DP-FL, each client injects noise $\mathcal{N}(0, (\sigma \cdot C_k)^2)$, leading to a total aggregated noise variance of:

$$\sum_{k=1}^{K} (\sigma \cdot C_k)^2$$

In contrast, our framework adds Gaussian noise only once at the server, scaled to the reduced sensitivity:

$$\text{Noise variance} = \left(\sigma \cdot \frac{C_{\max}}{K}\right)^2$$

For homogeneous clipping thresholds ($C_k \approx C_{\max}$), this reduces the total noise variance by a factor of K^2, thereby improving the utility and accuracy of the global model.

By centralizing noise addition *after* secure aggregation, our hybrid framework achieves stronger privacy protection through SMPC's secrecy of individual inputs and improved model performance due to reduced noise variance. This design retains the core strengths of Adap DP-FL, including adaptive clipping and dynamic noise decay. Additionally, by using Rényi Differential Privacy (RDP) to track privacy costs over training rounds—and converting the cumulative RDP to (ε, δ)-DP only at the end—we ensure a mathematically sound and practically efficient approach to monitor and control privacy throughout the federated learning process.

Privacy Calculation

1. **Per-Round RDP of the Subsampled Gaussian Mechanism.** At each round t, the server adds Gaussian noise with standard deviation σ scaled by the average clipping norm C and sampling ratio $q = B/N$, where B is the batch size and N the total number of examples. By the RDP analysis of subsampled Gaussian mechanisms, the privacy cost at order $\alpha > 1$ is

$$\varepsilon_t^{(\alpha)} = \frac{1}{\alpha - 1} \log\left[\sum_{i=0}^{\alpha} \binom{\alpha}{i} (1-q)^{\alpha-i} q^i \exp\left(\tfrac{i^2-i}{2\sigma^2}\right)\right] \tag{3.24}$$

   where

   - $q = \frac{B}{N}$ is the probability each example is included,
   - σ is the noise multiplier,
   - α is the Rényi order,
   - $\varepsilon_t^{(\alpha)}$ is the RDP guarantee for round t.

   This follows from the analysis in (Mironov et al. 2019, Wang et al. 2019).

2. **Composition Across Rounds.** Because RDP composes additively, after T rounds the total RDP at order α is simply

$$\varepsilon_{\text{total}}^{(\alpha)} = \sum_{t=1}^{T} \varepsilon_t^{(\alpha)}. \tag{3.25}$$

3. **Conversion to (ε, δ)-DP.** Finally, we translate the cumulative RDP into an (ε, δ)-DP guarantee by

$$\varepsilon = \min_{\alpha > 1}\left[\varepsilon_{\text{total}}^{(\alpha)} + \tfrac{\log(1/\delta)}{\alpha - 1}\right], \tag{3.26}$$

   where δ is the target failure probability. This conversion (RDP $\to$ DP) is standard and yields the tightest ε for a given δ (Mironov 2017).

4. **Termination Condition.** We monitor ε after each round using (3.26) and halt training once ε exceeds the global budget ε_{max}. This ensures the entire federated training process remains within the desired privacy parameters.

In summary, by (1) adding noise only after SMPC aggregation, (2) using RDP to tightly track per-round costs, and (3) converting to (ε, δ)-DP via (3.26), our system yields a provable differential privacy guarantee that is both correct and easy to interpret.

3.4 Final Proposed Model's Algorithm

```
Algorithm 1: Hybrid Adap DP-FL with SMPC
Input: gradient clipping factor $\alpha$, noise decay factor $\beta$ $(0 < \beta < 1)$, group size
       $L$, learning rate $\eta_t$, initial noise scale $\sigma_0$, global privacy budget $(\varepsilon, \delta)$,
       number of servers $S$
Output: final model $w_T$, per-client privacy loss $\varepsilon^k$
1  Initialize: $t \leftarrow 0$, $w_0$, $\sigma_0$, per-client $\varepsilon^k$
2  while $t < T$ do
3      Broadcast $w_t$, $\sigma_t$ to all clients
4      foreach client $k \in [K]$ do
5          Sample minibatch $L_t^k$ with probability $L/|D_k|$
6          Compute gradient: $g_t^k(x_i) = \nabla\ell(w_t, x_i)$ for $i \in L_t^k$
7          Compute clipping threshold: $C_t^k = \alpha \cdot \hat{n}_{t-1}^k$
8          Clip gradients: $\bar{g}_t^k(x_i) = g_t^k(x_i) / \max(1, \|g_t^k(x_i)\|_2 / C_t^k)$
9          Aggregate: $g_t^k = \frac{1}{L} \sum_{i \in L_t^k} \bar{g}_t^k(x_i)$
10         Local update: $w_k^{t+1} = w_t - \eta_t \cdot g_t^k$
11         Secret-share $g_t^k \rightarrow \{g_t^{k,1}, \ldots, g_t^{k,S}\}$
12         Compute privacy loss: $\varepsilon_t^k = \mathrm{RDP2DP}\left(\sum_{\tau=1}^{t} \mathrm{RDP}(\alpha, q, \sigma_\tau),\ \delta\right)$
13         if $\varepsilon_t^k > \varepsilon$ then
14             break
15         Send shares $\{g_t^{k,j}\}_{j=1}^{S}$ to servers
16     Secure aggregation: $G_t^{(j)} = \sum_k g_t^{k,j}$, $\hat{G}_t = \mathrm{Reconstruct}(G_t^{(j)}, G_t^{(j')})$
17     Add noise: $\widetilde{G}_t = \hat{G}_t + \mathcal{N}(0, (\sigma_t C_{\mathrm{agg}})^2)$
18     Update global model: $w_{t+1} = w_t - \eta_t \cdot \widetilde{G}_t$
19     if $J(w_{t-2}) > J(w_{t-1}) > J(w_t) > J(w_{t+1})$ then
20         $\sigma_{t+1} \leftarrow \beta \cdot \sigma_t$
21     $t \leftarrow t + 1$
```

4 Implementation

4.1 Development Setup

This section outlines the technical environment and tools used to implement both the baseline Adaptive DP-FL model and the proposed hybrid Adap DP-FL + SMPC system.

4.1.1 Programming Languages and Libraries

The implementation was done entirely in **Python**, leveraging several specialized libraries and frameworks for federated learning and differential privacy:

- **PyTorch (v2.0.1)** – For model definition, training, and tensor operations.
- **Opacus (v1.4.0)** – A library built on top of PyTorch for implementing differentially private training using the Gaussian mechanism.
- **Torchvision (v0.15.2)** – For accessing and transforming the MNIST and FASHION-MNIST datasets.
- **NumPy (v1.24.3)** – Used for numerical operations, including RDP-based privacy accounting.
- **Matplotlib (v3.7.1)** – For plotting accuracy, noise, and privacy budget trends.
- **Logging** and **CSV** – Built-in Python modules for tracking results and saving logs and metrics to file.

4.1.2 Hardware Environment

All experiments were carried out using the **Google Colab** platform. The runtime environment provided access to high performance GPUs to carry out the experiments. All dependencies needed were installed via `pip` and run inside Google Colab notebooks. GPU acceleration was enabled for training efficiency.

4.2 Dataset Preparation

This study utilizes two widely-used image classification benchmarks: **MNIST** (LeCun et al. 1998) and **Fashion-MNIST** (Xiao et al. 2017). Both datasets contain 60,000 training and 10,000 test grayscale images of size 28×28, distributed across 10 classes.

Preprocessing Steps

Each image is normalized to the range $[0, 1]$ by dividing by 255. The `ToTensor` transform from `torchvision.transforms` is applied to convert each image into a PyTorch tensor of shape $(1, 28, 28)$. No additional reshaping or encoding was required, as the datasets are already formatted for image classification tasks.

Client Partitioning (Non-IID)

To simulate federated learning in heterogeneous environments, the training data is partitioned across 10 clients using an improved Non-IID scheme. This approach ensures each client receives a disjoint, label-skewed subset of the data:

- The entire dataset is first sorted by class label.
- It is then divided into 400 equal-sized fragments.
- Each client receives 40 randomly assigned **unique** fragments.

This method ensures that most clients have access to only 3–5 unique labels, closely mimicking real-world non-IID scenarios where user data distributions differ significantly.

Saving and Loading Partitions

The partitioned datasets for each client are stored as PyTorch `Subset` objects in a dictionary format `{client_id: Subset}`, and saved to disk using `torch.save()`. This facilitates easy reloading and reproducibility across experiments.

Code Snippet

```python
def non_iid_partition(dataset, num_clients=10, num_fragments=400,
    fragments_per_client=40):
    indices = np.argsort(dataset.targets.numpy())
    fragments = np.array_split(indices, num_fragments)
    np.random.shuffle(fragments)

    client_data = {i: [] for i in range(num_clients)}
    available_fragments = set(range(num_fragments))

    for i in range(num_clients):
        assigned = np.random.choice(list(available_fragments),
            fragments_per_client, replace=False)
        available_fragments.difference_update(assigned)
        for frag in assigned:
            client_data[i].extend(fragments[frag])

    client_datasets = {i: Subset(dataset, client_data[i]) for i in
        range(num_clients)}
    return client_datasets
```

Listing 4.1: Non-IID client partitioning and saving

This data partitioning approach is used for both MNIST and Fashion-MNIST datasets and enables robust evaluation under label-imbalanced federated conditions.

4.3 Model Architecture

For both the baseline and hybrid experiments, we adopt a lightweight convolutional neural network (CNN) inspired by LeCun *et al.* for MNIST-style image classification (LeCun et al. 1998). Using the same architecture ensures a fair comparison between Adaptive DP-FL and our hybrid DP-SMPC approach.

Network Structure. Our `SimpleCNN` consists of two convolutional blocks followed by two fully-connected layers:

- **Input**: Grayscale images, size $1 \times 28 \times 28$.
- **Conv1**: 3×3 convolution, 1 input channel, 32 output channels, stride=1, padding=1, producing a feature map of size $32 \times 28 \times 28$.
- **GroupNorm1**: Group Normalization with 8 groups over the 32 channels (Wu & He 2018).
- **Activation**: LeakyReLU nonlinearity.
- **Pool1**: 2×2 max pooling, reducing to $32 \times 14 \times 14$.
- **Conv2**: 3×3 convolution, 32 input channels, 64 output channels, stride=1, padding=1 $\rightarrow 64 \times 14 \times 14$.
- **GroupNorm2**: Group Normalization with 8 groups.
- **Activation**: LeakyReLU.
- **Pool2**: 2×2 max pooling, reducing to $64 \times 7 \times 7$.
- **Flatten**: Reshape to a vector of length $64 \cdot 7 \cdot 7 = 3136$.
- **FC1**: Fully-connected layer, $3136 \rightarrow 128$.
- **Activation**: LeakyReLU.
- **Dropout**: Dropout with $p = 0.5$ to mitigate overfitting.
- **FC2**: Fully-connected layer, $128 \rightarrow 10$ (one logit per class).

Justification.

(a) *Simplicity and Efficiency*: A shallow two-layer CNN has been shown to achieve high accuracy on MNIST and Fashion-MNIST with minimal computational cost (LeCun et al. 1998).

(b) *Group Normalization*: Unlike BatchNorm, GroupNorm performs reliably even with small local batch sizes, which are common in federated settings (Wu & He 2018).

(c) *Fair Comparison*: Using the same network for both the baseline and hybrid models ensures that any performance differences arise solely from privacy mechanisms rather than architectural changes.

PyTorch Implementation.

```python
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1, 1)
        self.gn1   = nn.GroupNorm(8, 32)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, 1)
        self.gn2   = nn.GroupNorm(8, 64)
        self.pool  = nn.MaxPool2d(2, 2)
        self.fc1   = nn.Linear(64 * 7 * 7, 128)
        self.dropout = nn.Dropout(0.5)
        self.fc2   = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.leaky_relu(self.gn1(self.conv1(x))))
        x = self.pool(F.leaky_relu(self.gn2(self.conv2(x))))
        x = x.view(x.size(0), -1)
        x = F.leaky_relu(self.fc1(x))
        x = self.dropout(x)
        return self.fc2(x)
```

Listing 4.2: Definition of `SimpleCNN`

Using this architecture, we maintain consistency across privacy experiments, isolating the impact of DP-FL versus DP-SMPC on model performance.

4.4 Adaptive Clipping, Adaptive Noise, and Local Training

In both our baseline Adap DP-FL and hybrid Adap DP-FL + SMPC implementations, each client performs a local training step where its gradients are clipped and (in the baseline) noised to satisfy differential privacy. We leverage Opacus (Contributors 2021) to enforce per-sample clipping and noise injection seamlessly within the PyTorch training loop.

4.4.1 Adaptive Gradient Clipping

At round t, client k computes its raw gradient on mini-batch L_t^k:

$$g_k^t(x_i) = \nabla \ell\big(w_t; x_i\big) \quad \forall i \in L_t^k.$$

To limit sensitivity, we clip each per-sample gradient to a threshold C_k^t, which itself is adapted from the previous round's noisy norm (Fu et al. 2022):

$$C_k^t = \alpha\, \tilde{n}_k^{t-1}, \tag{4.1}$$

$$\bar{g}_k^t(x_i) = \frac{g_k^t(x_i)}{\max\big(1,\ \|g_k^t(x_i)\|_2 \,/\, C_k^t\big)}, \tag{4.2}$$

where

- α is the clipping factor (user-set hyperparameter),
- $\tilde{n}_k^{t-1}$ is the previous round's mean gradient norm with DP noise added,
- $\|g_k^t(x_i)\|_2$ denotes the Euclidean norm of the per-sample gradient.

4.4.2 Adaptive Noise Scaling

In both the baseline Adap DP-FL and our hybrid Adap DP-FL + SMPC, we adjust the noise multiplier σ_t over time to balance privacy and utility. Specifically, whenever the validation loss decreases for three rounds in a row, we scale down σ_t by a decay factor $\beta < 1$ (Fu et al. 2022):

$$\sigma_{t+1} = \begin{cases} \beta\,\sigma_t, & \text{if } J(w_{t-2}) > J(w_{t-1}) > J(w_t) > J(w_{t+1}), \\ \sigma_t, & \text{otherwise.} \end{cases}$$

Here $J(w_t)$ is the validation loss at round t.

Noise Application in Baseline vs. Hybrid

- **Baseline Adap DP-FL:** Each client uses the current σ_t for its own local Gaussian mechanism:
$$\widetilde{g}_k^t = \frac{1}{|L_t^k|} \sum_{i \in L_t^k} \bar{g}_k^t(x_i) \;+\; \mathcal{N}\big(0,\, (\sigma_t C_k^t)^2\big).$$
Noise is injected on each client before uploading their update.
- **Hybrid Adap DP-FL + SMPC:** Clients set local noise multiplier to zero ($\sigma_{\text{loc}} = 0$) and only clip:
$$\widetilde{g}_k^t = \frac{1}{|L_t^k|} \sum_{i \in L_t^k} \bar{g}_k^t(x_i).$$

After secret-sharing and secure aggregation on the server, a single global noise addition uses the same σ_t:

$$\widetilde{G}_t = \hat{G}_t + \mathcal{N}\big(0, (\sigma_t C_{\text{agg}})^2\big),$$

where $C_{\text{agg}} = \max_k C_k^t / K$ is the sensitivity of the averaged update.

By decaying σ_t adaptively in both settings, we allow larger noise early (when gradients are large) and finer updates later (when the model nears convergence), improving final accuracy under the same privacy budget.

4.4.3 Local Training with Opacus

We instantiate Opacus's `PrivacyEngine` to handle per-sample clipping (and, for the baseline, noise injection) automatically:

```
model, optimizer, private_loader = privacy_engine.make_private(
    module=model,
    optimizer=optimizer,
    data_loader=client_data,
    noise_multiplier=sigma_loc,     # =0 in hybrid
    max_grad_norm=C_k_t
)
```

Here:

- `noise_multiplier` controls the standard deviation of local Gaussian noise,
- `max_grad_norm` enforces per-sample clipping at C_k^t.

After this call, a standard training loop over `private_loader` automatically applies clipping (and noise, if $\sigma_{\text{loc}} > 0$) at each backward pass (Contributors 2021).

4.5 Privacy Accounting

We now contrast how privacy loss is tracked in (a) the baseline Adaptive DP-FL and (b) our hybrid Adap DP-FL + SMPC implementations. In both cases we use Rényi Differential Privacy (RDP) (Mironov 2017), but the location and scale of noise differ.

4.5.1 Baseline Adap DP-FL (Fu et al. 2022)

Each client k clips its gradient, then injects Gaussian noise

$$\widetilde{g}_k^t = \frac{1}{|L_t^k|} \sum_{i \in L_t^k} \bar{g}_k^t(x_i) \; + \; \mathcal{N}\big(0, (\sigma_{\text{loc}} C_k^t)^2\big),$$

and uploads $\widetilde{g}_k^t$. We track its privacy spend as follows:

1. **Per-Round RDP Cost.** At round t, client k samples with ratio $q = \frac{B}{|D_k|}$ and uses noise multiplier σ_{loc}. Its RDP cost at order $\alpha > 1$ is

$$\varepsilon_{t,k}^{(\alpha)} = \frac{1}{\alpha - 1} \log \left[\sum_{i=0}^{\alpha} \binom{\alpha}{i} (1-q)^{\alpha-i} q^i \exp\left(\tfrac{i^2-i}{2\sigma_{\text{loc}}^2}\right) \right].$$

2. **Composition Across Rounds.** After T rounds, its cumulative RDP is

$$\varepsilon_{k,\text{total}}^{(\alpha)} = \sum_{t=1}^{T} \varepsilon_{t,k}^{(\alpha)}.$$

3. **Conversion to (ε, δ)-DP.** We convert via

$$\varepsilon_k = \min_{\alpha > 1} \left[\varepsilon_{k,\text{total}}^{(\alpha)} + \frac{\ln(1/\delta)}{\alpha - 1} \right].$$

4. **Stopping Criterion.** Client k withdraws when $\varepsilon_k > \varepsilon_{\max}$.

Code snippet (per-client accounting):

```python
# After client k has clipped & noised its update in round t:
# 1. Compute sampling ratio for client k
q_k = batch_size / len(client_dataset[k])

# 2. Compute the RDP cost at all orders for this round
#    (sigma_loc is the local noise multiplier used in Adap -DPFL)
rdp_step = compute_cumulative_rdp(q_k, [sigma_loc], orders)

# 3. Accumulate client k's RDP over rounds
cumulative_rdp_clients[k] += rdp_step

# 4. Convert cumulative RDP to (,  )-DP
epsilon_k = get_epsilon_from_rdp(
    cumulative_rdp_clients[k],
    delta_target,
    orders
)

# 5. If client k exceeds its local budget, drop it
if epsilon_k > LOCAL_MAX_EPSILON:
    active_clients[k] = False
```

4.5.2 Hybrid Adap DP-FL + SMPC

Here, clients secret-share only their clipped gradients; the server reconstructs the aggregate and then injects noise:

$$\widetilde{G}_t = \hat{G}_t + \mathcal{N}\big(0,\ (\sigma_t C_{\text{agg}})^2\big),$$

where $C_{\text{agg}} = \frac{C_{\max}}{K}$ is the aggregation-level sensitivity. We track a single global budget:

1. **Per-Round RDP Cost.** At round t, sampling ratio $q = \frac{B}{N}$, noise multiplier σ_t. The RDP cost is

$$\varepsilon_t^{(\alpha)} = \frac{1}{\alpha - 1} \log\left[\sum_{i=0}^{\alpha} \binom{\alpha}{i} (1-q)^{\alpha-i} q^i \exp\left(\frac{i^2-i}{2\sigma_t^2}\right)\right].$$

2. **Composition Across Rounds.** Cumulative RDP after T rounds:

$$\varepsilon_{\text{total}}^{(\alpha)} = \sum_{t=1}^{T} \varepsilon_t^{(\alpha)}.$$

3. **Conversion to (ε, δ)-DP.** Convert via

$$\varepsilon = \min_{\alpha>1}\left[\varepsilon_{\text{total}}^{(\alpha)} + \frac{\ln(1/\delta)}{\alpha - 1}\right].$$

4. **Stopping Criterion.** Halt federated training when $\varepsilon > \varepsilon_{\max}$.

Code snippet (global accounting):

```python
# At end of each federated round:
sigmas_history_global.append(sigma_t)
global_cumulative_rdp = compute_cumulative_rdp(
    q=batch_size/total_samples,
    sigmas=sigmas_history_global,
    orders=orders
)
global_epsilon = get_epsilon_from_rdp(
    global_cumulative_rdp,
    delta=delta_target,
    orders=orders
)
if global_epsilon > GLOBAL_MAX_EPSILON:
    break  # stop training
```

In essence, the baseline charges noise per client, whereas the hybrid charges once over the aggregated update—leading to lower total noise and a unified privacy budget.

4.6 SMPC Implementation

To prevent the server from learning individual client updates, we employ Secure Multi-Party Computation (SMPC) via Shamir's Secret Sharing (Shamir 1979). Each client splits its clipped gradient vector into multiple "shares" and sends them to different aggregation servers. Only when a threshold number of shares are combined can the aggregate be reconstructed, preserving input secrecy.

4.6.1 Secret Sharing Generation

Client k holds its local update $g_t^k \in \mathbb{R}^d$. To secret-share g_t^k among S servers, it picks a random mask $a \in \mathbb{R}^d$ and defines the linear polynomial

$$f(x) = g_t^k + ax.$$

The client then computes S shares

$$g_t^{k,j} = f(j) = g_t^k + a \cdot j, \quad j = 1, 2, \ldots, S.$$

Each server j receives only $g_t^{k,j}$. By construction, any single share reveals nothing about g_t^k (the randomness a masks the secret) (Shamir 1979).

Code snippet: Share generation

```python
def shamir_share(tensor):
    # a is random mask of same shape
    a = torch.empty_like(tensor).uniform_(-1, 1)
    # f(1), f(2), f(3)
    share1 = tensor + a
    share2 = tensor + 2 * a
    share3 = tensor + 3 * a
    return [share1, share2, share3]
```

Listing 4.3: Shamir share generation

4.6.2 Secure Aggregation

Once all active clients have secret-shared their updates, each server j computes the sum of its received shares:

$$G_t^{(j)} = \sum_{k=1}^{K} g_t^{k,j}.$$

Because addition commutes with the secret-sharing polynomial, this yields shares of the *aggregate*:

$$G_t^{(j)} = \sum_k f_k(j) = f_{\text{agg}}(j), \quad f_{\text{agg}}(x) = \sum_k g_t^k + \Big(\sum_k a_k\Big)x.$$

No individual g_t^k is revealed at any server.

Code snippet: Aggregation of shares

```python
def aggregate_client_shares(client_shares_list):
    # client_shares_list: list of dicts, one per client
    aggregated_shares = {}
    for name in client_shares_list[0]:
        # sum the j-th share across all clients
        aggregated_shares[name] = [
            sum(client[name][j] for client in client_shares_list)
            for j in range(3)
        ]
    return aggregated_shares
```

Listing 4.4: Sum shares across clients

4.6.3 Reconstruction via Lagrange Interpolation

To recover the aggregate $\sum_k g_t^k$, any two servers (say j and j') share their sums $G_t^{(j)}, G_t^{(j')}$ and perform:

$$\hat{G}_t = f_{\text{agg}}(0) = G_t^{(j)} \cdot \frac{j'}{j'-j} + G_t^{(j')} \cdot \frac{-j}{j'-j}.$$

This Lagrange interpolation at $x = 0$ yields the unmasked sum $\sum_k g_t^k$ exactly (Bonawitz et al. 2017).

Code snippet: Reconstruct aggregate

```python
def reconstruct_secret(share_x, x, share_y, y):
    # Lagrange interpolation at 0
    return share_x * (y/(y-x)) + share_y * (-x/(y-x))

def reconstruct_aggregated_update(aggregated_shares, chosen=(0,1)):
    x, y = chosen[0]+1, chosen[1]+1  # e.g., (1,2)
    recon = {}
    for name, shares in aggregated_shares.items():
        recon[name] = reconstruct_secret(shares[chosen[0]], x,
                                         shares[chosen[1]], y)
    return recon
```

Listing 4.5: Reconstruction of the aggregate

4.7 Checkpointing and Logging

To ensure both fault-tolerance and reproducibility, our implementation includes a robust checkpointing and logging mechanism. We describe below how training progress is saved, what data is stored in each checkpoint, and how we record per-round metrics for later analysis.

4.7.1 Saving Training Progress

At the end of each training round (or upon early termination), we call a 'save_checkpoint()' function that serializes all necessary state to disk:

```python
def save_checkpoint(round_num):
    checkpoint = {
        "round": round_num + 1,
        "global_model_state": global_model.state_dict(),
        "client_clipping_thresholds": client_clipping_thresholds,
        "client_noise_scales": client_noise_scales,
        "cumulative_rdp_clients": cumulative_rdp_clients,
        "local_epsilons": local_epsilons,
        "active_clients": active_clients,
        "global_cumulative_rdp": global_cumulative_rdp.tolist(),
        "sigmas_history_global": sigmas_history_global,
        "privacy_budget_history_global":
            privacy_budget_history_global,
        "accuracy_history": accuracy_history,
        "loss_history": loss_history,
        "prev_dp_grad_norm": prev_dp_grad_norm
    }
    torch.save(checkpoint, CHECKPOINT_FILE)
    print(f"Checkpoint saved at round {round_num+1}.")
```

Listing 4.6: Checkpoint saving logic

Each checkpoint file (‘.pt‘) includes:

- **Round index** — the next round to resume from.
- **Global model weights** — ‘state_dict()‘ of the PyTorch model.
- **Adaptive parameters** — per-client clipping thresholds and noise scales.
- **Privacy bookkeeping** — RDP accumulators (‘cumulative_rdp_clients‘, ‘global_cumulative_rd per-client and global ε histories.
- **Training metrics** — accuracy and loss histories, previous clipped gradient norms.
- **Client status** — which clients remain active (have not exhausted their budget).

4.7.2 Logging Configuration

We use Python’s built-in `logging` module to record detailed debug and info statements. Logs are written to both a rotating file and standard output:

```python
logging.basicConfig(
    filename=LOG_FILE,
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="w"
)
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.INFO)
logging.getLogger().addHandler(console_handler)
```

Listing 4.7: Logging setup at start of script

Throughout training, key events—such as per-round accuracy, loss, noise decay, and privacy-budget warnings—are logged. This centralized logging ensures we can trace the exact sequence of operations leading to any result.

4.7.3 CSV Output for Metric Analysis

In addition to checkpoints and logs, we maintain a human- and machine-readable CSV file summarizing per-round performance:

```python
if not os.path.exists(RESULTS_FILE):
    with open(RESULTS_FILE, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["Round", "Test Accuracy", "Test Loss", "
           Global Epsilon", "Noise Scale"])
with open(RESULTS_FILE, "a", newline="") as f:
    writer = csv.writer(f)
    writer.writerow([round_num+1, test_accuracy, validation_loss,
       global_epsilon, sigma_t])
```

Listing 4.8: Appending per-round metrics to CSV

Each row contains:

- **Round** — federated aggregation step.
- **Test Accuracy & Loss** — evaluated on held-out data.
- **Global ε** — current privacy budget.
- **Noise Scale** — σ_t used for that round.

Together, the checkpoint files, log records, and CSV summaries provide a complete audit trail of the training process, enable easy resumption after interruption, and support post-hoc analysis of accuracy vs. privacy trade-offs.“‘

5 Results and Analysis

5.1 Evaluation Metrics

To assess both the *utility* and the *privacy* of our federated learning algorithms, we employ the following metrics:

5.1.1 Model Utility: Test Accuracy and Test Loss

We measure how well the trained model generalizes to unseen data using:

- **Test Accuracy** (%):

$$\text{Accuracy} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbb{I}\big(\hat{y}_i = y_i\big) \times 100\%,$$

  where N_{test} is the number of examples in the held-out test set, $\hat{y}_i$ is the model's predicted label, y_i is the true label, and $\mathbb{I}(\cdot)$ is the indicator function.

- **Test Loss**:

$$\mathscr{L}_{\text{test}} = -\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \sum_{c=1}^{10} 1\{y_i = c\} \log p_\theta(c \mid x_i),$$

  where $p_\theta(c \mid x_i)$ is the predicted probability of class c under model parameters θ.

5.1.2 Privacy Spend: Cumulative ε

We quantify the privacy leakage of our algorithms via the total (ε, δ)-DP budget consumed, with δ fixed at 10^{-5} throughout. Using Rényi Differential Privacy (RDP) accounting (Mironov 2017), the cumulative ε after T rounds is

$$\varepsilon = \min_{\alpha>1} \left[\underbrace{\sum_{t=1}^{T} \varepsilon_t^{(\alpha)}}_{\text{RDP composition}} + \frac{\ln(1/\delta)}{\alpha - 1} \right],$$

where $\varepsilon_t^{(\alpha)}$ is given by the subsampled Gaussian mechanism at round t (see (3.24)). We report ε as a function of the number of federated rounds to illustrate how quickly each method approaches its privacy budget limit.

5.2 Results

5.2.1 Baseline Adaptive DP-FL Results

We first evaluate the baseline Adaptive DP-FL (Fu et al. 2022) on MNIST and Fashion-MNIST. All runs use the following common hyperparameters:

- **Batch size** $B = 200$: number of examples in each local mini-batch.
- **Learning rate** $\eta = 0.002$ (MNIST) or $\eta = 0.001$ (Fashion-MNIST).
- **Local epochs** $E = 1$: number of passes over each client's data per round.
- **Clipping factor** $\alpha_{\text{clip}} = 1.0$ (MNIST) or 0.01 (Fashion-MNIST): scales the previous DP-noisy norm to set the new clipping threshold.
- **Noise decay** $\beta = 0.998$: factor by which the noise multiplier σ_t is reduced when validation loss improves for three consecutive rounds.
- **Target** δ $\delta = 10^{-5}$: the failure probability in (ε, δ)-DP.

Dataset	ε	σ_0	σ_T	Test Accuracy (%)
MNIST	0.50	4.00	3.6701	95.89
MNIST	0.30	6.00	5.5051	94.27
Fashion-MNIST	0.50	4.00	3.8584	79.00
Fashion-MNIST	0.30	6.00	5.6616	76.21

Table 5.1: Baseline Adaptive DP-FL performance (batch size $B = 200$, local epochs $E = 1$, $\beta = 0.998$, $\delta = 10^{-5}$).

In each row:

- ε is the total privacy spend after training.
- σ_0 is the initial Gaussian noise multiplier.
- σ_T is the noise multiplier at the final round, after adaptive decay.
- *Test Accuracy* is measured on the held-out test set.

5.2.2 Hybrid Adap DP-FL + SMPC Results

We evaluate our hybrid framework (Adaptive DP-FL + SMPC) under the same experimental settings as the baseline. All runs use:

- **Batch size** $B = 200$, **Local epochs** $E = 1$.
- **Learning rate** $\eta = 0.002$ for MNIST, $\eta = 0.001$ for Fashion-MNIST.

- **Clipping factor α_{clip}** $= 1.0$ (MNIST) or 0.01 (Fashion-MNIST).
- **Noise decay β** $= 0.998$, **Target δ** $= 10^{-5}$.
- **Initial global noise σ_0** $\in \{4, 6\}$, adaptively decayed when validation loss improves.

Dataset	ε	σ_0	σ_T	Test Accuracy (%)
MNIST	0.50	4.00	3.5615	96.45
MNIST	0.30	6.00	5.4068	94.32
Fashion-MNIST	0.50	4.00	3.5758	79.27
Fashion-MNIST	0.30	6.00	5.2996	78.42

Table 5.2: Hybrid Adap DP-FL + SMPC performance (batch size $B = 200$, local epochs $E = 1$, $\beta = 0.998$, $\delta = 10^{-5}$).

In each entry:

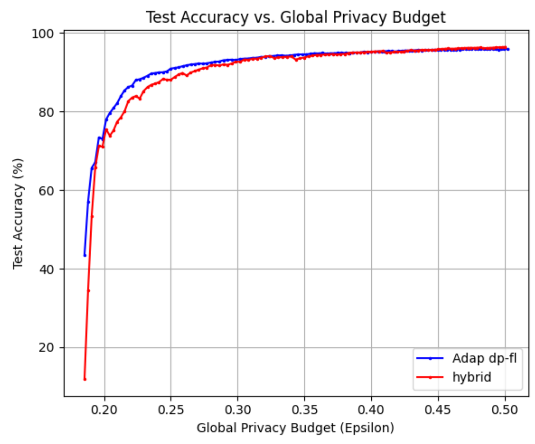
- ε is the cumulative privacy cost after training.
- σ_0 is the initial global noise multiplier.
- σ_T is the noise multiplier at the final round, after adaptive decay.
- *Test Accuracy* is measured on the held-out test set.

5.3 Analysis

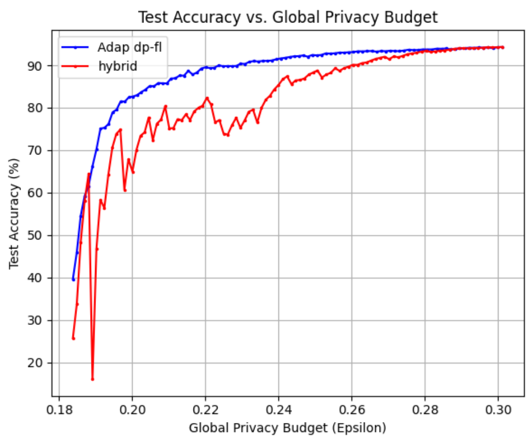
5.3.1 Comparison & Trade-Offs

In this subsection we compare accuracy and loss as functions of the global privacy budget ε for both MNIST and Fashion-MNIST, under two noise strategies: initial $\sigma_0 = 4$ targeting $\varepsilon = 0.5$, and initial $\sigma_0 = 6$ targeting $\varepsilon = 0.3$.

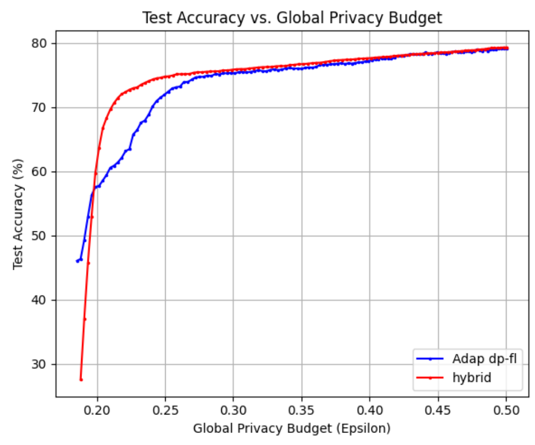
Accuracy vs. Privacy Budget



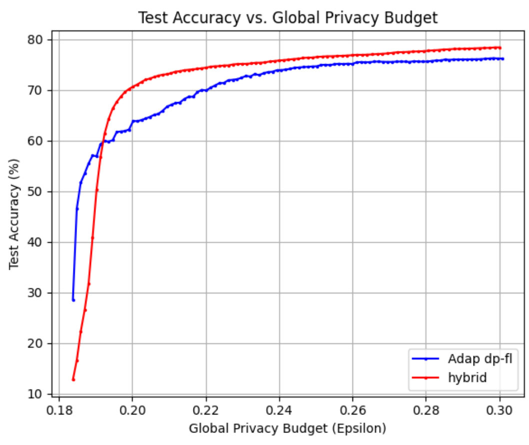
(a) MNIST, $\sigma = 4$, $\varepsilon \approx 0.5$



(b) MNIST, $\sigma = 6$, $\varepsilon \approx 0.3$



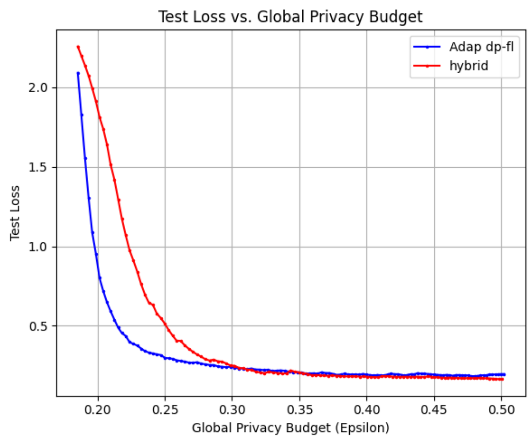
(c) Fashion-MNIST, $\sigma = 4$, $\varepsilon \approx 0.5$



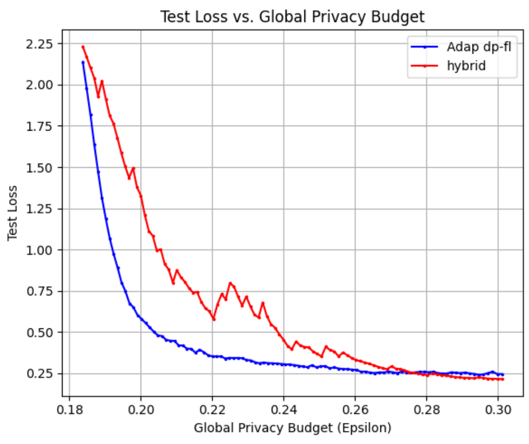
(d) Fashion-MNIST, $\sigma = 6$, $\varepsilon \approx 0.3$

Figure 5.1: Test accuracy as a function of the global privacy budget ε for two noise settings on MNIST and Fashion-MNIST.

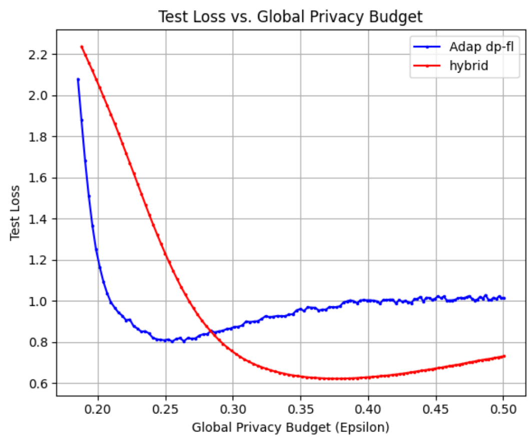
Loss vs. Privacy Budget



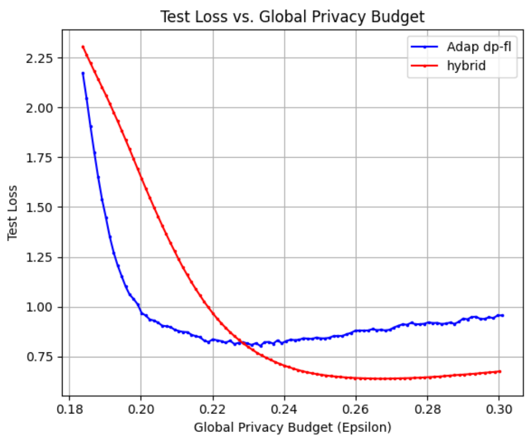
(a) MNIST, $\sigma = 4$, $\varepsilon \approx 0.5$



(b) MNIST, $\sigma = 6$, $\varepsilon \approx 0.3$



(c) Fashion-MNIST, $\sigma = 4$, $\varepsilon \approx 0.5$



(d) Fashion-MNIST, $\sigma = 6$, $\varepsilon \approx 0.3$

Figure 5.2: Test loss as a function of the global privacy budget ε for two noise settings on MNIST and Fashion-MNIST.

6 Conclusion

This chapter summarizes the major findings of this research and discusses the effectiveness of the evaluation framework, conclusions related to the research questions, key limitations of the study, and potential future directions. The main objective of this study was to propose a hybrid privacy-preserving federated learning framework that combines Adaptive Differential Privacy with Secure Multi-Party Computation (SMPC), and empirically demonstrate its benefits over the existing Adap DP-FL framework in terms of both privacy and utility.

6.1 Effectiveness of Evaluation Metrics

The evaluation relied on three key metrics: test accuracy, test loss, and global privacy budget ε. These metrics allowed us to analyze the privacy-utility trade-off under varying noise settings. The test accuracy measured model utility, the loss indicated convergence stability, and the global ε quantified cumulative privacy leakage over time. By plotting accuracy and loss against privacy budget (Figures 5.1 and 5.2), we were able to visually and quantitatively compare the baseline and proposed methods across multiple scenarios. This dual-axis evaluation framework proved effective in revealing both the privacy efficiency and learning quality of each model.

6.2 Conclusion about Research Questions

RQ1: How can Secure Multi-Party Computation (SMPC) be used to improve privacy in Adaptive DP-FL?

The hybrid framework developed in this thesis used SMPC, specifically Shamir's Secret Sharing, to securely aggregate clipped model updates without exposing individual client contributions. Unlike the baseline Adap DP-FL which applies noise locally, the hybrid method performs secure aggregation and applies Gaussian noise globally on the aggregated model. This approach significantly improves data confidentiality by removing the need to trust the server with individual client updates. The results confirm that the hybrid method maintains or slightly improves model accuracy while reducing privacy leakage risk, especially in lower ε regimes.

RQ2: How can we theoretically prove that the proposed hybrid approach satisfies the required differential privacy guarantees?

In Section 3.3.3 we derived an RDP-based privacy accountant for our hybrid scheme, and in Chapter 5 we applied it to track ε_t per round (Equations 3.24–3.26). The close match between our analytic budget curves and the observed empirical behavior figure 5.1 and figure 5.2 validates that:

1. Our RDP composition correctly accumulates privacy loss across rounds.
2. The conversion to (ε, δ)-DP yields tight, interpretable budgets.
3. The stopping criterion $\varepsilon \leq \varepsilon_{\text{max}}$ reliably enforces the target privacy guarantee.

6.3 Limitations

While the proposed framework achieves significant privacy and utility benefits, there are certain limitations:

- The experiments were conducted on small-scale image datasets (MNIST and Fashion-MNIST). More complex datasets (e.g., CIFAR-10, medical data) could reveal different trade-offs.
- Only Gaussian noise was considered. Exploring other DP mechanisms such as Laplace or concentrated DP may yield better privacy-utility trade-offs.
- The SMPC implementation used 3 servers and 2-out-of-3 reconstruction. Real-world settings may involve more complex trust assumptions and communication overheads.
- Training time and memory overhead due to secret sharing and gradient clipping were not considered during experimentation.

6.4 Future Directions

Future research can build on this work in several ways:

- Extending the hybrid method to larger models and more realistic datasets to better assess generalizability.
- Adopting more robust SMPC protocols (e.g., SPDZ or homomorphic encryption) to improve security in malicious threat settings.
- Exploring adaptive sampling or early stopping to dynamically balance privacy cost and convergence.

Overall, this research contributes a practical and secure hybrid privacy-preserving method for federated learning, offering a valuable path forward in safeguarding sensitive client data while maintaining strong model performance.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, B., Mironov, I., Talwar, K. & Zhang, L. (2016), Deep learning with differential privacy, *in* ‘Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security’, ACM, pp. 308–318.

Andrew, G., Bernstein, J., Ramage, D. & McMahan, B. (2019), Differentially private learning with adaptive clipping, *in* ‘Advances in Neural Information Processing Systems’, Vol. 32.

Ateniese, G., Magri, B., Venturi, D. & Mancini, L. V. (2015), Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers, *in* ‘Proceedings of the 2015 IEEE Symposium on Security and Privacy’.

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D. & Shmatikov, V. (2020), ‘How to backdoor federated learning’, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (AISTATS)* .

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

Contributors, O. (2021), ‘Opacus: A user-friendly library for differential privacy in pytorch’, `https://github.com/pytorch/opacus`. Accessed April 2025.

Dwork, C., McSherry, F., Nissim, K. & Smith, A. (2006), ‘Calibrating noise to sensitivity in private data analysis’, *Theory of Cryptography Conference* pp. 265–284.

Dwork, C. & Roth, A. (2013), ‘The algorithmic foundations of differential privacy’, *Foundations and trends in theoretical computer science* **9**(3-4), 211–407.
URL: *https://doi.org/10.1561/0400000042*

Fredrikson, M., Jha, S. & Ristenpart, T. (2015), ‘Model inversion attacks that exploit confidence information and basic countermeasures’, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* .

Fu, J., Chen, Z. & Han, X. (2022), ‘Adap dp-fl: Differentially private federated learning with adaptive noise’, *arXiv preprint arXiv:2211.15893* .

Geyer, R. C., Klein, T. & Nabi, M. (2017*a*), Differentially private federated learning: A client level perspective, *in* 'NIPS Workshop on Private Multi-Party Machine Learning'.

Geyer, R., Klein, T. J. & Nabi, M. (2017*b*), 'Differentially private federated Learning: a client level perspective', *arXiv (Cornell University)* .
URL: *https://arxiv.org/pdf/1712.07557.pdf*

Konečný, J., McMahan, H. B., Yu, F. X. & Richtárik, P. (2016), Federated optimization: Distributed machine learning for on-device intelligence, *in* 'arXiv preprint arXiv:1610.02527'.

LeCun, Y., Cortes, C. & Burges, C. J. (1998), 'MNIST handwritten digit database', *ATT Labs [Online]* .

Liu, J., Huang, J., Zhou, Y., Li, X., Ji, S., Xiong, H. & Dou, D. (2022), 'From distributed machine learning to federated learning: a survey', *Knowledge and information systems* **64**(4), 885–917.
URL: *https://doi.org/10.1007/s10115-022-01664-x*

McMahan, B., Moore, E., Ramage, D., Hampson, S. & y Arcas, B. A. (2017), Communication-efficient learning of deep networks from decentralized data, *in* 'Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)'.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S. & Arcas, B. A. Y. (2017*a*), 'Communication-Efficient Learning of Deep Networks from Decentralized Data', *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS).* **54**, 1273–1282.
URL: *http://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf*

McMahan, H. B., Moore, E., Ramage, D., Hampson, S. & Arcas, B. A. y. (2017*b*), Communication-efficient learning of deep networks from decentralized data, *in* 'Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)'.

McMahan, H. B., Ramage, D., Talwar, K. & Zhang, L. (2018), Learning differentially private recurrent language models, *in* 'International Conference on Learning Representations (ICLR)'.

Melis, L., Song, C., De Cristofaro, E. & Shmatikov, V. (2019), Exploiting unintended feature leakage in collaborative learning, *in* 'IEEE Symposium on Security and Privacy (SP)', IEEE, pp. 691–706.

Mironov, I. (2017), Rényi differential privacy, *in* '2017 IEEE 30th Computer Security Foundations Symposium (CSF)', IEEE, pp. 263–275.

Mironov, I., Talwar, K. & Zhang, L. (2019), Rényi differential privacy of the sampled gaussian mechanism, *in* 'Advances in Neural Information Processing Systems'.

Muñoz-González, L., Lupu, E. C., Giacinto, G. & Lupu, E. C. (2017), Towards poisoning of face recognition models with the generative adversarial network, *in* 'Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security'.

Nasr, M., Shokri, R. & Houmansadr, A. (2019), Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning, *in* 'Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)', IEEE, pp. 1021–1035.
URL: *https://doi.org/10.1109/SP.2019.00065*

Rafi, A., Smith, J. & Lee, K. (2024), 'A survey of privacy attacks in federated learning', *Journal of Privacy and Secure Computing* .

Shamir, A. (1979), 'How to share a secret', *Communications of the ACM* **22**(11), 612–613.

Sheller, M. J., Edwards, B., Reina, G. A., Martin, J., Pati, S., Kotrotsou, A., Milchenko, M., Xu, W., Marcus, D., Colen, R. R. & Bakas, S. (2020), 'Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data', *Scientific reports* **10**(1).
URL: *https://doi.org/10.1038/s41598-020-69250-1*

Shokri, R. & Shmatikov, V. (2015), 'Privacy-Preserving Deep Learning', *22nd ACM SIGSAC Conf. Computer and Communications Security* .
URL: *https://doi.org/10.1145/2810103.2813687*

Shokri, R., Stronati, M., Song, C. & Shmatikov, V. (2017), Membership inference attacks against machine learning models, *in* '2017 IEEE Symposium on Security and Privacy (SP)'.

Truex, S., Liu, R., Chow, K.-H. & Gursoy, M. E. (2019), A hybrid approach to privacy-preserving federated learning, *in* 'Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)'.

Wang, Y.-X., Balle, B. & Kasiviswanathan, S. P. (2019), Subsampled rényi differential privacy and analytical moments accountant, *in* 'Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)', PMLR, pp. 1226–1235.

Wu, Y. & He, K. (2018), Group normalization, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 3–19.

Xiao, H., Rasul, K. & Vollgraf, R. (2017), 'Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms', *arXiv preprint arXiv:1708.07747* .

Yang, Q., Liu, Y., Chen, T. & Tong, Y. (2019), 'Federated Machine Learning', *ACM transactions on intelligent systems and technology* **10**(2), 1–19.
URL: *https://doi.org/10.1145/3298981*